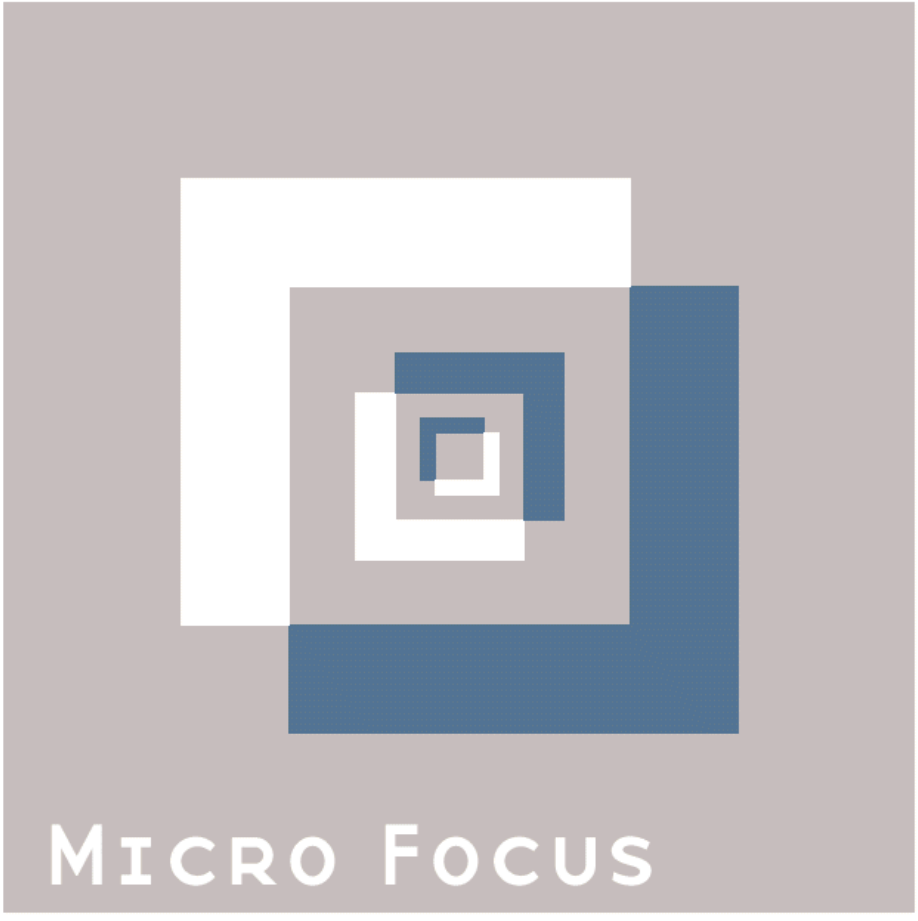
Issue 2, November 2001



APS FOR ZOS

USER'S GUIDE

MICRO FOCUS

20020219093933

Table of Contents

1 Introduction

This chapter contains the following sections:

Introduction to APS

Generate MIS applications automatically

APS for z/OS is a full-function application generator that automates the development and redevelopment of the MIS applications that support your business. With APS you can build simple or complex applications for a variety of IBM SAA production environments. You can generate online and batch applications without manual coding.

Improve application quality

APS improves both the quality of your applications and your efficiency in developing them. Quality improves because you focus on the end user's functional requirements, and not the application's physical implementation. As a result, applications generated with APS are more likely to meet user expectations without extensive modifications.

APS lets developers focus on user needs by working in a variety of development cycles. For example, if your approach is:

- Rapid Application Development (RAD), you use APS to prototype your application user interface and its technical environment. You generate a working application directly from the prototype, without wasting any steps.
- Waterfall, you develop requirements, specifications, and designs in a front-end CASE tool such as Excelerator. In this analysis- and design-driven development, you then transfer that information to APS, where it becomes the basis of the design you use to generate the working application.

- Hybrid, you can start either with prototypes or high-level specifications, and move freely between the two approaches as you refine your models. APS gives you the flexibility to adapt your lifecycle approach to particular project requirements.
- Redevelopment, you can capture high-level information about your existing applications, and then forward engineer applications using APS. APS makes it easy to maintain or enhance applications by reusing design information that defines application screens, databases, and other features.

Improve development productivity

Productivity improves because you can generate a complete working application without first becoming an expert in IMS, CICS, ISPF, SQL, or any other environment that APS supports. Because APS lets you focus on high-level requirements and specifications, novice developers can quickly generate simple designs with minimal training, and soon build incrementally to complex applications.

APS also encourages developers to share and reuse design modules. It lets you store all application design information in one central location on a network or mainframe, so that multiple users can access that information concurrently. As a result, your application designs always reflect the entire team's current work, and it is easy to share data.

Build your knowledge base into APS

APS lets developers share design components that leverage the knowledge and experience of your most senior people. Expert COBOL developers can customize APS so that it supports your organization's requirements for such features as user interface, report writing, and naming conventions. When these senior APS users define in-house standards or solve complex problems, you can incorporate their work into the dynamic APS rules base, where it is available to all users at your site.

Because APS lets you work from high-level designs, you can easily retarget APS applications for multiple environments. You simply specify the new target, and regenerate the application, using the same programs, screens, and other application components. APS lets you use your existing database subschemas, tables, and files in the new applications you generate.

A Scenario for Using APS

Start development with tasks you prefer

APS lets you work in whatever sequence you choose. For example, you can define the user interface before you think about the program logic. Or you can first define the global data that all programs must use. Whether you decide to work top down, bottom up, or middle out, APS lets you proceed from the step you just completed to the one you want to do next. And, if you are maintaining or redeveloping existing applications, APS lets you focus on only those components that must change.

Build applications from high-level specifications

It's often best to begin by defining your application's runtime or target environment. When you select an environment, APS handles the necessary implementation details when it generates the application. APS for z/OS supports the following targets:

Database	DB/2 IMS DB VSAM IDMS
Data Communication	CICS IMS DC ISPF ISPF Dialog MVS (batch)

Generate applications for multiple targets

If your application must run in several target environments, you can easily specify it to do so. For example, you can generate an application so that it can access more than one kind of database. In each case, APS generates code that runs in the environments you specify in your high-level application design.

Whenever you are ready, you import into APS the subschemas or tables for your existing databases. Your APS applications can then access these existing databases--first for prototype testing within APS, and then for running the final application.

Prototype the application's look and feel

You can next prototype the look and feel of an application, so that end users can review it early in the design process. Then paint the screens that support the user interface you select.

Paint applications

APS lets you paint menu and data entry screens that include data entry and text fields. You can also include message fields that let the application communicate errors.

At this stage of prototyping, you and your end users run the screens to ensure that the application user interface meets end user expectations. You can enter data into data entry fields to ensure that they capture all of the required information. You can test various display sequences of screens to ensure that the screens support an intuitive work flow. Because it is so easy to create, run, and change these prototypes, you can work closely with end users to refine this aspect of the application before you move on to design the underlying details of the application.

Specify application logic

APS also simplifies the task of defining an application's processing logic. You can generate online applications from high-level designs using the default logic that APS produces, or you can tailor the logic to your particular requirements. For example, you can tailor the way that APS processes database calls, error routines, or other program functions.

You create batch applications using whatever combination of specification language, user-defined macros, or COBOL/2 syntax that you prefer. You can combine online and batch programs as you like within a single application.

Automatically generate a working application

When you generate an application, it is ready to install and set up to run in your production environment. Generation produces consistent, high-quality code without run-time modules.

Test run the application

Once you generate an application, you can test run your work within APS using the APS ISPF Prototype Execution facility. This facility emulates the basic functions of the mainframe CICS or IMS DC environments, letting you test the data communication and database functions of your programs. Doing so allows you to find features that do not meet expectations, and then modify and retest those features without first setting up the complete target environment that the application will ultimately run in.

Using this facility, you can test out your application's program navigation and flow--for example, sending screens, passing control from one program to another, and terminating programs. If you have imported your database definitions, you can test the prototype using test data in your actual SQL or VSAM (but not IMS) database; otherwise you can test the processing logic using data that you enter into screens but do not store.

The APS Tool Set

Basic APS painters and facilities

The basic APS toolset lets you generate applications from high-level specifications. The APS tools that support this work are as follows:

- Application Painter lets you specify the target environment. You also use the Application Painter to name the components of the working application, such as its programs; screens or report mockups; and data structures, and to define the relationships among them.
- Database Importers let you use existing databases as part of any APS application.
- Screen Editor lets you paint screens containing global or local data elements.
- Scenario Painter lets you prototype the flow and behavior of screens.
- Online Express lets you define all program logic by using predefined teleprocessing and database function codes, mapping record fields to screens, defining and qualifying database calls, and adding your own program functions to support specialized requirements.
- Specification Painter lets you extend Online Express applications with customized processing logic that supports your coding practices.
- Program Painter is an alternative to Online Express, and lets you create batch and online applications using COBOL II and high-level APS constructs such as database calls, data communication calls, and Report Writer constructs.
- Generators let you create an executable COBOL application from your high-level APS specification.
- Documentation Facility lets you produce reports about your applications.

Advanced APS painters and facilities

If you need to customize your applications to reflect specific in-house coding practices, APS also provides a set of advanced facilities that give you this flexibility. These tools are as follows:

- Data Structure Painter lets you define data elements you can reuse in applications.

- Data Element Facility lets APS Administrators set and enforce in-house standards by creating and maintaining global data elements that all developers use in their applications.
- User Help Facility lets you create online help systems for your applications.
- Customization Facility lets you extend APS applications using macros you define to support your coding practices.
- APS/ENDEVOR Interface lets you link to LEGENT's ENDEVOR software management product to manage the different versions of your application components.

2 Paint the Application Definition

This chapter contains the following sections:

- *Application Painter Concepts*
- *Painting an Application Definition*
- *Defining Application Components*

Application Painter Concepts

List application components

The APS Application Painter lets you define your application by listing all of its components in a matrix. The matrix provides both an overview of the entire application, and easy access to the other APS painters and facilities where you define or import the application components.

An application can include the following components:

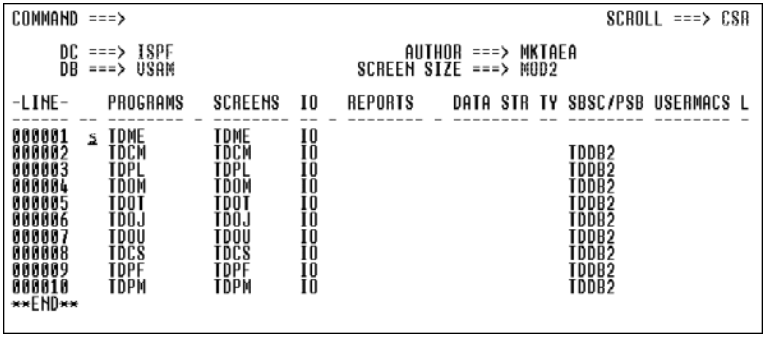
- A combination of online and batch programs
- User interface screens
- Report mock-ups
- Data structures
- Subschemas and PSBs
- User-defined macros
- Subroutines, called global program stubs

These components can define an application of whatever scope you require. For example, one application might be an entire Order Inventory system, while another might be an Order Status database inquiry.

List components as they relate to each other

You list application components in the Application Painter screen so that the matrix indicates their relationships. To do so, you type a program name and the names of all the components that belong with the program on one row. If you want multiple programs to share components such as subschemas, data structures, or user-defined macros, you type these component names on one or more rows above your first program. Shared components are known as global components; all programs in your application can reference them.

Figure 2-1. A Sample Application Definition



For example, you first type the name of any global subschema that some or all of your programs access. On the next row, you type the name of the first program and its associated screen. Then you type the other program and screen names on subsequent rows. If you want one of the programs to access a subschema other than the global subschema, you type its name next to that program. If one of your programs is a batch report program, you type the name of its report mock-up, and indicate that the program is a batch program.

Define and generate components

Once you name these components, you can use the matrix to navigate to the other APS facilities where you develop the components. As you complete individual programs or the application as a whole, you can return to the Application Painter to generate them into executable COBOL source or to generate reports on the progress of your work.

Target your database and data communications environment

As part of the application definition, you specify your target environment--the environment where you want your application to run. APS generates your application to run in the database/data communications (DB/DC) environment you specify. You can generate

your application for another environment simply by changing your DB/DC target specification.

You can write an application that consists entirely of online programs, entirely of batch programs, or you can mix online programs with batch programs in the same application. In addition, an application--or a single program--can access multiple DB/DC targets. For example, your online programs can use CICS to access VSAM files and SQL databases, while your batch programs access VSAM files and IMS databases.

Develop and test your application as a prototype

The Application Painter also supports the prototyping that speeds your development work. This painter lets you access the Scenario Painter, where you can test your application's behavior without having to access your database. For example, you can simulate executing your application, to determine whether its screens display in the sequence you want. You can reorder the sequence as desired, without leaving the Scenario Painter. In addition, the Scenario Painter lets you enter sample data on your screens to test how they accept and display data.

When you are ready to access your database, you can test your application from within APS, using the APS Prototype Execution facility. This facility emulates your production CICS or IMS environment.

Painting an Application Definition

To define an application, list components on the Application Painter screen as follows:

Display Application Painter screen

1. From the APS Main Menu, enter 1 in the Command field. The Painter Menu displays.
2. To access the Application Painter screen, enter e(dit) in the Command field, ap(plication) in the Type field, and the application name in the Member field. The application name can be eight characters maximum; the first character must be alphabetic; others can be alphanumeric or special characters.

Specify DC target

3 Specify the data communications (DC) target in the DC field, as described below:

If application contains . . .	Specify this DC target . . .
Only online programs	Your online DC target, such as CICS. For a list of valid DB/DC combinations for generating executable programs to run on various operating systems, see the "DB/DC Target Combinations" topic in the APS Reference.
Only batch programs	Mvs. Additionally, leave each Screen field and I/O field blank.
Both online and batch programs	Your online DC target. To identify programs as batch, enter *batch in the Screens field next to each batch program name and leave the I/O fields blank.

Specify DB target

4 Specify the database (DB) target in the DB field. For a list of valid DB/DC combinations, see the "DB/DC Target Combinations" topic in the APS Reference. To specify a SQL target, leave the DB field blank or let default to VSAM. Then go to the Generator Options screen and specify the SQL target.

Note: If your application accesses multiple database targets, specify your DB target as follows:

If application accesses ...	Specify this DB target ...
Two DB targets, including VSAM	The non-VSAM target; APS always gives you access to the VSAM target.
Two or more DB targets, not including VSAM	Any of those DB targets. When you generate the programs, first generate the programs of the specified DB target. Then change the DB target to the next target and generate the programs of that target. For example, if your application accesses both SQL and IMS subschemas, generate the SQL programs separately from the IMS programs.

Note: Specify your target operating system when you prepare to generate the application.

Prototype using ISPF

5 If you are creating a CICS or IMS DC application that accesses SQL or VSAM databases and you want to create a prototype of the application, you can execute and test within the APS Prototype Execution facility. Set the DC target to ISPF and the DB target to SQL or VSAM. After testing the ISPF prototype, change the DB/DC targets to the production targets and regenerate the application.

Specify screen size

6 Specify the size of the screen for your application. Enter one of the following application screen sizes in the Screen Size field. Ensure that the development screen lets you create application screens of the size you want as follows:

Application Screen Size	Dimension	Development Screen Size
MOD2	24 x 80 lines	MOD2, MOD3, MOD4, or MOD5
MOD3	32 x 80 lines	MOD3 or MOD4
MOD4	43 x 80 lines	MOD4
MOD5	27 x 132 lines	MOD5

Specify program

7 Enter your first online or batch program name in the Programs field. The name can be eight characters maximum. The first character must be alphabetic; others can be alphabetic, numeric, or the special characters #, $, or @. The names all and dummy are invalid.

Specify screen

8 For online programs, enter the program associated screen name in the Screens field, on the same row as the program name. Adhere to the following naming conventions:

- CICS screen names can be seven characters maximum. The first character must be alphabetic; others can be alphanumeric.
- IMS screen names can be eight characters maximum. The first character must be alphabetic; others can be alphanumeric.
- ISPF Dialog screen names can be eight characters maximum.
- ISPF prototype screen names can be seven characters maximum.

For batch programs, enter *batch in the Screens field, on the same row as the program name.

Specify screen I/O

9 On the same row as your first screen name, use the IO field to specify whether the screen is input-only (i), output-only (o), or input/output (io). For batch programs, leave the IO field blank.

Specify report mock-up

10 To specify a batch program's report mock-up, enter the mock-up name in the Reports field. The name can be eight characters maximum. The first character must be alphabetic or the special characters #, $, or @; others can be any of these or numeric. You create a mock-up using the APS Report Painter. For information, see *Create Reports with Report Writer*.

Specify data structure

11 Still on the same row, specify the name of any data structure file that the program will reference. The name can be eight characters maximum. The first character must be alphabetic; others can be alphanumeric. To make the data structure global, or available to all programs of the application, enter its name on a row above all programs.

You create a data structure using the APS Data Structure Painter. For information, see the "Data Structures" topic in the APS Reference.

12 If you specified a data structure file, specify in the Ty(pe) field the program location where you plan to include it:

WS Working-Storage Section

LK Linkage Section

CA Program Commarea

Specify subschema or PSB

13 Enter your program subschema or PSB name in the Sbsc/PSB field. The name can be eight characters maximum. The first character must be alphabetic; others can be alphanumeric. To make the subschema or PSB global, or available to all programs of the application, enter its name on a row above all programs.

You import your existing subschema or PSB into APS using the APS Importer Facility. See *Import Database Definitions* for information.

Specify user-defined macro library member

14 Still on the same row, specify any user-defined macro library member that this program will reference. Enter the name in the USERMACS field. The member that you specify must reside in your Project and Group's USERMACS data set. The name can be eight characters maximum. The first character must be alphabetic; others

can be alphanumeric. To make the member global, or available to all programs of the application, enter its name on a row above all programs.

You create macros using the APS Customization Facility language structures. For information, see the ***APS Customization Facility User's Guide***.

15 If you specified a macro library member, specify in the Loc(ation) field the program location where you plan to invoke its macros. Valid location values are as follows:

Location Code	Description
T	Default; top of program, before Identification Division
B	Bottom of program
WT	Top of Working-Storage Section
WS	Working-Storage Section, after any data structures you include in the Data Str field
WB	Bottom of Working-Storage Section
LT	Top of Linkage Section
LK	Linkage Section, after any data structures you include in the Data Str field
LB	Bottom of Linkage Section
IO	Top of Input-Output Section
FD	Top of File Section
RP	Top of Report Section
CA	Top of Commarea

Specify global stub

16 To include procedural subroutines that all programs of the application can reference, known as global stubs, enter on a separate row the stub name in the Programs field, and enter *stub in the Screens field. The name can be eight characters maximum. The first character must be alphabetic; others can be alphanumeric or the special characters #, $, or @. Regardless of the row where you enter a global stub name, any program of the application can reference it.

17 On subsequent rows, specify the rest of your programs and their associated components, following steps 7 through 16.

18 To insert, move, copy, and delete rows of the application definition, use the ISPF commands: insert; move; copy; delete; before; after.

19 Save the application definition by pressing PF3 or entering save in the Command field. You can modify it at any time.

Special Considerations

- To create a new application definition quickly, you can copy an existing one and modify it. To do so, use the Create Like function on the Painter Menu.
- Deleting a component from the Application Painter matrix removes it from the application definition, but not from the APS Dictionary. This component is available to add it to other applications. However, if you delete a component from the Painter Menu, you remove it from the APS Dictionary, and must separately delete it from any other applications that reference it.

Defining Application Components

To complete the application, you define each component using other APS painters and facilities, following these steps, in any sequence you want:

1 Position the cursor in the selection field to the left of the component you want to define. Then enter one of the following selection codes to access the painter you want:

   - Online program, enter ox to access Online Express, a nonprocedural, menu-driven facility for quickly defining online COBOL-based programs. For details, see *Define Processing Logic* and *Define Database Access*.
   - Batch program, enter s to access the Program Painter, where you can use APS database calls, data communications calls, and Report Writer structures to speed batch program writing. For details, see *Create Batch Programs*, and *Create Reports with Report Writer*.

- Screen, enter s to access the the Screen Painter (if you are creating a character-based application). For details, see *Paint Character Screens*.
- Report mock-up, enter s to access the APS Report Mock-up Painter, where you define the physical layout of reports. For details, see *Create Reports with Report Writer*.
- Data structure, enter s to access the Data Structure Painter where you define Working-Storage data elements. For details, see the "Data Structures" topic in the APS Reference.
- Global program stub, enter s to access the Program Painter, where you define Procedure Division paragraphs to customize or supplement APS-generated program logic. For details, see *Define Processing Logic*.

2 Alternatively, access the painter or facility you want from the Painter Menu. To do so, specify one of the following types in the Type field: ds (Data Structure Painter), pg (Program Painter), rp (Report Mock-up Painter), or sc (Screen Painter). Then enter the component name in the Member field, or press Enter to display a list of components to select from.

3 To make your subschemas or PSBs available to your application, you import them into APS using the APS database importers. The importers generate your SQL and IDMS subschemas, IMS PSBs and DBDs, and VSAM files into a format usable with your APS programs. For details, see *Import Database Definitions*.

4 To create user-defined macros that provide program logic to meet your own site-specific requirements, use any text editor to create macros and store them in the USERMACS library in your APS Project and Group. For details, see the APS Customization Facility User's Guide.

3 Import Database Definitions

This chapter contains the following sections:

Importer Concepts

Transfer database definitions to APS

APS Import Facilities allow you to transfer information about your database definitions and their copybook records to APS programs. You can import:

- IMS DBDs and PSBs
- IDMS subschemas
- SQL DB2 objects
- VSAM files

You can also use APS Import Facilities to transfer BMS and MFS screens. For more information about importing screens, see *Paint Character Screens*.

Translate database information

The APS Import Facilities translate database information such as data definitions and/or subschemas into a format usable for generating and precompiling through APS. When the APS Import Facility transfers database information, the Database Definition Interface (DDI) formats the database information to use with APS programs.

Combine multiple databases

If required you can combine multiple database environments into a single subschema by giving each subschema the same name when you

import it using the APS database importers. Then simply reference that name in any application that requires it.

The database importers:

- Extract information from your database definition.
- Load extracted information into the DDIFILE.
- Generate a DDISYMB file for use by Online Express and the appropriate APS Generator.
- Generate record description copybooks of SQL DDL statements that contain database and COBOL descriptions of each table or view in the imported subschema.

Importing IMS PSBs and DBDs

Code DDI statements before you import

Before you import IMS database definitions, you must code DDI statements to identify which IMS segments and COBOL copylib record descriptions to import. You can import the following IMS database information:.

Input	Library	Description
PSB	*project.group*.PSBSRC	Native PSB source; no modification is necessary.
DBD	*project.group*.DBDSRC	Native DBD source; no modification is necessary.
copylibs	*project.group*.COPYLIB	For each IMS segment in your DBD, you must have a copylib containing a COBOL record description.

Code DDI statements and import IMS database information as follows:

1 Copy the PSB and DBD into *project.group*.PSBSRC and DBDSRC files respectively, and specify the PSB on the Application Painter. For more information on the Application Painter, see *Paint the Application Definition*.

2 Copy into *project.group*.COPYLIB one or more COBOL copylib files containing COBOL record descriptions for each IMS segment in your DBD.

3 Code a DDI DBD statement to correspond to the DBD statement in your database. For syntax information, see the "DDI Statements" topic in the APS Reference.

4 Code a DDI REC statement for each segment you want to import to correspond to the SEGM statement in your database and its copylib as follows:

a Specify the name of the copylib record that corresponds to the DBD segment identified on the DDI DBD statement using the NAME parameter. The name of the copylib member that contains the segment copylib record should be the same as the segment name. If it is not, specify the copylib member name with the COPY parameter.

b Set the &GEN-DB-REC-01 NAMES flag in the APS CNTL file, APSDBDC, to indicates the level number of your top level copylib records.

- If your top level copylib records begin with 01, set flag to 0.
- If your top-level copylib records do not begin with 01, set flag to 1 and assign a unique 01-level name using the NAME parameter. Alternately, use the GEN01 parameter to override the value of this flag. Specify GEN01=n to indicate that the top level copylib record level number begins with an 01 level number or specify GEN01=y to indicate that the top level numbers do not begin with 01.

  For example, if your copylib record looks like this:

```
05  WS-EMPLOYEE-INFO.
    10  WS-EMPLOYEE-NO              PIC 9 (06).
```

  Code the DDI REC statement as:

```
DDI REC NAME=WS-EMPLOYEE-STUFF, SEG=EMPLSEG,
COPY=EMPLDATA
```

  The generated output would look like this:

```
01  WS-EMPLOYEE-STUFF.
    05  WS-EMPLOYEE-INFO.
      10  WS-EMPLOYEE-NO             PIC 9 (06).
```

For syntax information, see the "DDI Statements" topic in the APS Reference.

5 Code a DDI FLD statement to correspond to the copylib record field and each field statement in the DBD. If your database contains secondary indexes, you can search on a secondary index field without having to generate DDISYMB for the index database. For syntax information, see the "DDI Statements" topic in the APS Reference.

- If the XDFLD has multiple SRCH fields, do one of the following to include SRCH fields:
  - Code a DDI FLD statement. When there are multiple SRCH fields, APS defaults the name value to the XDFLD value. Code a dummy COBOL name that is unique from any copylib field name of the SRCH fields in the NAME parameter to override the default.
  - Specify the DBD value of XDFLD using the IMSNAME parameter.

    For example, for the following DBD source:

    ```
    XDFLD NAME=SOUCOX2, SRCH= (EMPLASNA,SOURCODE)
    ```

    Code the DDI FLD statement as follows:

    ```
    *DDI FLD NAME=INDEX-NO2,IMSNAME=SOUCOX2
    ```

    Hint: Use the dummy name you specified on the DDI FLD statement in database commands to qualify on a secondary index composed of multiple SRCH fields.
- If the XDFLD has one SRCH field, code DDI DBD and DDI REC statements only. It is not necessary to code a DDI FLD statement for the XDFLD because the APS Generator refers to the SRCH field definition. Note: If you write a database command against a PCB that uses a secondary index, use the proper COBOL name for the index field to be qualified upon. The APS Generator recognizes a secondary index by the presence of the PROCSEQ or INDICES parameters, and generates segment search arguments (SSA) naming the IMS XDFLD.

Code DDI for logical relationships

6 If necessary, you can code DDI statements for IMS logical DBDs and PSBs that reference IMS logical relationships as follows:

a Code DDI DBD statement.

b Code a DDI REC statement. Do not include information for segments in a logical DBD if the logical segment has a single physical source segment or the same IMS name as its physical source segment. Note: The APS/IMS Generator does not validate IMS logical Insert/Delete/Replace rules.

7 Place DDI statements in *project.group.*DDISRC.

8 Access the IMS Database Importer. To do so, from the APS Main Menu, enter 2, Dictionary Services in the Command field. On the Dictionary Services screen, enter 1, Import Facilities in the Command field. Enter 2, IMS on the Import Facilities screen.

9 On the IMS Importer screen, type the DBD member name in the Member field and enter 1, Load DBD Definitions and *DDI statements, and generate DBD in the Option field. APS reads, extracts and stores the DBD and DDI statement information in the DDIFILE.

10 After option 1 completes, enter 2, Load PSB definitions and Generate PSB and DDISYMB in the Option field. This option references the information in the DDIFILE and reads, extracts, and translates PSB information into DDI symbols and stores it in *project.group*.DDISYMB.

Supplementing or Overriding DDI Statements

When you import a DBD, you can write DDI statements that assign a set of COBOL record and field names to each segment and field in the PSB. Depending on what you want your program to do, you can supplement or override these names with additional DDI statements. For example, if you must maintain multiple positioning on a segment type, use more than one PCB to reference the same segment thereby maintaining multiple positioning. You need at least two areas in Working-Storage in which to retrieve the same segment. This way the retrieval of one segment will not overlay the Working-Storage area of the same segment retrieved at a different position in the database. Another example is multiple programs that reference the same segment type, but some programs must use different record descriptions of the segment. In this case, you can override the names defined in the original DDI statements on a program by program basis.

To supplement or override record and field names write additional DDI statements in a separate DDI statement member. Modify JCL in the APS PSB utility member and reimport the database to include the new DDI statements. To do so, follow the steps below.

1 Create a member in *project.group*.DDISRC dataset giving it the same name as the program PSB.

2 Write DDI statements to assign the new set of names to the segment. Use the format below, starting each statement in column 7.

```
-KYWD-      12-*----20----*----30---*----40----*----50---
*--
 *DDI    PSB NAME=psbname
 *DDI    PCB
 *DDI    PCB
 *DDI    PCB
 .
 .
 .
 *DDI    REC SEG=segmentname,NAME=new-COBOL-recordname,
 COPY=new-copylibname
 *DDI    FLD IMSNAME=name,NAME=new-COBOL-fieldname
 *DDI    FLD IMSNAME=name,NAME=new-COBOL-fieldname
 *DDI    FLD IMSNAME=name,NAME=new-COBOL-fieldname
```

The DDI statements are described below:

DDI Statement	Description
*DDI PSB	Specifies the program PSB.
*DDI PCB	Positional or placeholder statements that indicate the PCB for which you are assigning an additional set of names. For example, to assign names to the fourth PCB in the PCB, write four *DDI PCB statements; do not write a *DDI PSB statement for any subsequent PSB.
*DDI REC	Specifies the following: • Segment name as it appears in the program PSB. • New COBOL record name of the segment. • New copylib name for the new COBOL record.

DDI Statement	Description
*DDI FLD	Specifies the following: • IMS field name as it appears in the program PSB. • New COBOL name of a field in the new COBOL record. Write one statement per field.

3 Copy the APS PSB utility member, &APSPRE..ISPSLIB(SSMXPSB), to the dataset that is concatenated before &APSPRE..ISPSLIB.

4 Modify the copy of SSMXPSB as follows:

- Add the parameter parm='ddi' to the //DDIIMS statement so that it reads as follows:

```
//DDIIMS     EXEC PGM=DDIIMS,REGION=1024k,PARM='DDI',
          COND=((0,LT,PSBGEN),(O,LT,LINK))
```

- Change the //DDICARDS DD DUMMY statement to the following:

```
//DDICARD    DD  DISP=SHR,DSN=&DDIPRE..DDISRC(&SSMDDI)
```

5 Select option 2.1.2 to display the APS IMS Database Importer. Enter the program DBD name in the Member field and execute option 1, Load DBD Definitions and DDI Statements, and Generate DBD.

6 After option 1 completes, execute option 2, Load PSB Definitions and Generate PSB and DDISYMB.

7 Return to the READY prompt and restart APS.

Importing SQL DB2 Objects

Import DB2 object to the common data area

The SQL Importer lets you import DB2 objects stored in the DB2 system catalog to a separate staging area in APS. This staging area, known as the APS common data area, is where you create and generate subschemas for imported DB2 objects using the SQL Subschema Maintenance utilities. For more information about these utilities, see the Administrator's Guide: Chapter 3, "Managing APS Facilities and Libraries."

You can import the following DB2 object types using the SQL Importer:

- Alias (object only)
- Database
- Index (object only)
- Storage group
- Table and table space
- View (object only)

Import DB2 objects as follows:

1. From the APS Main Menu enter 2, in the Option field. APS displays the Dictionary Services screen. Enter 1 in the Option field. APS displays the Import Facilities screen. Enter 1 in the Option field. APS displays the SQL Importer screen. Enter 1 in the Option field to display the DB2/DBP Object Import screen.
2. In the Command field, type the number that corresponds to the object type you want to import.
3. Type the name of the object you want to import in the Object Name field. If you do not know the name of the object, leave this field blank and press Enter. The SQL Importer displays the DB2 Object List screen. This screen displays data set information on the object types in the DB2 system catalog.
4. Select objects from the list by typing an s to the left of each desired object name. You can scroll the screen using the ISPF UPnn and DOWNnn commands and by setting the SCROLL field to page, half or csr. Selecting an object creates utility control cards that direct the batch job to import the object to the common data area. To cancel your selections and return to the DB2/DBP Object Import screen, type cancel or can in the Command field and press Enter.
5. Type end in the Command field and press Enter or press the appropriate PF key to view a screen that lists the objects you selected from the object list. The SQL Importer displays the list of selected objects. To delete a selection from this list, type d in the selection field.

6 Return to the DB2/DBP Object Import screen by doing one of the following:

- Press Enter
- Type end in the Command field and press Enter
- Press the appropriate PF key

7 Complete the remaining fields on the DB2/DBP Object Import screen as described below and press Enter.

Field	Value	
Object Creator	If you typed a name in the Object name field, type the object creator's TSO ID. Defaults to your TSO ID.	
Job Class	Type the job submission class. Valid values are:	
	J1-J5	APS defines job cards J1-J5 on the Job control cards screen. To access this screen, type J in this field.
	JC	ISPF job card defined on the ISPF Log andLists Defaults (0.2). You must increment the job card letter.
Object Only	Y(es)	Default for Index, View and, Alias. Imports the object specified in the Command field or selected from the Object List screen without associated objects.
	N(o)	Default for Storage Group, Data Base, Table Space, Table, and Column. Imports the object specified in the Command field or selected from the Object List screen plus all objects associated with the specified objects.
Report Only	Y(es)	Generates an import report for the object specified but does not import the object.
	N(o)	Default. Generates an import report in addition to importing the objects specified. For more information about reports, see *Generating a DB2/DBP Object Import Report*.

Field	Value	
Submit Job Now	Y(es)	Default. Submits a batch job to import the object specified in the Command field or selected from the Object List screen.
	N(o)	Generates the import job JCL and stores it in the data set specified in the following fields. This JCL can be used for later job submission.

Generate DDISYMB symbols

8 Access the APS Generator Options screen. From the APS Main Menu, enter 0 in the Option field. APS displays the Options Menu. Enter 1 in the Option field. APS displays the Generator Options screen. Set the SQL field to a valid SQL target; for example, DB2, SQLDS, or SQL400. .

9 Access the SQL Importer screen and type 2 in the Option field.

10 The DDIFILE project and group for the DDISYMBs defaults to your current user project. Ensure that your current project is identical to the project and group under which the subschema was created.

11 Type the subschema name in the Member field and press Enter.

Note: For more information on setting generator options, see *Setting Options*.

Generating a DB2/DBP Object Import Report

Determine the impact of importing DB2 objects

You can determine the impact of importing a DB2 object to the common data area by generating a DB2/DBP Object Import report. You can generate this report before, during or after importing DB2 objects. Reports generated after the import reflect all additions, deletions, or name changes made to the dependent common data area objects at the time of report generation.

The DB2/DBP Object report provides a cross reference of the objects in the DB2 catalog and the APS common data area. It illustrates where a DB2 object fits into the hierarchy of the DB2 system catalog versus

where it fits into the hierarchy of the common data area. The cross reference report lists objects in groups to illustrate hierarchical dependency.

The DB2/DBP Object Import report can consist of:

- Two side-by-side comparison lists with the following information:
  - The imported DB2 object and the DB2 system catalog objects dependent upon it (those lower in the hierarchy).
  - The imported object plus common data area objects that are dependent upon it after it is imported.
- Two side-by-side comparison lists of the DB2 and common area objects that contain the imported object (those higher in the hierarchy).
- A list of the associated common data area objects added since the last import of the object type.

Special Considerations

- If your DB2 system supports referential integrity (DB2 version 2 and higher), the SQL Importer also imports tables referenced by foreign keys.
- When you generate an Object Import report, enter information in all required fields of the DB2/DBP Object Import screen.
- The dependent common data area objects are listed under the report column titled DBP Object.

Importing VSAM Files

Code DDI statements before importing

Before you import VSAM files, you must code DDI statements. The DDI statements identify the VSAM file and the COBOL copylib record descriptions you want to import. For each VSAM file record, you must have a copylib file that contains a COBOL record description. These copylib files must reside in *project.group*.COPYLIB.

Code DDI statements and import your VSAM file information as follows:

1. Copy the copylib file(s) that contain the COBOL record descriptions for the VSAM file(s) into *project.group*.COPYLIB COBOL.
2. Code a DDI VSM statement to correspond to the VSAM file external ddname. This statement identifies VSAM file attributes. You can specify parameters to generate IDCAMS. For syntax information, see the "DDI Statements" topic in the APS Reference.
3. Code a DDI REC statement for each copylib record to correspond to the copylib record name and copylib filename with the longest MAXLEN. For syntax information, see the "DDI Statements" topic in the APS Reference.
4. If your top-level copylib records do not begin with 01, set global flags &VS-GEN-01-USING-RECNAMES and GEN-DB-REC-01-NAMES and GEN-DB-REC-01-NAMESin APS CNTL files APVSAMIN and APSDBDC to yes.
5. If the VSAM file you want to import is keyed, code a DDI IDX statement for each index that corresponds to keyed copylib field name and copylib file name. All DDI IDX statements must immediately follow the DDI REC statement. Write overlapping record keys as ordinary DDI IDX statements. APS generates IDCAMS KEYS keyword according to the OFFSET and KEYLEN keywords on the DDI IDX statement. For syntax information, see the "DDI Statements" topic in the APS .
6. Code a DDI SUB statement to correspond to the copylib record name(s) and VSAM file external ddname. This statement defines a subschema for your VSAM file. Note: To define a subschema with multiple VSAM files, assign a RECORD keyword to each VSAM file you include. For syntax information, see the "DDI Statements" topic in the APS .
7. Assign a unique subschema name and enter it in the Application Painter field, SBSC/PSB.
8. Place the DDI statements in *project.group*.DDISRC.
9. Access the VSAM Importer. From the APS Main Menu screen, enter 2, Dictionary Services in the Option field then enter 1, Import Facility then enter 3, VSAM.
10. From the APS/VSAM File Importer screen, enter the DDISRC name (the member name) in the member field, and enter 1 Load DDI

Information From DDISRC in the option field. APS reads, extracts and stores the DDI statement information in the file DDIFILE.

11 After option 1 completes, execute option 3 - Generate DDISYMB Symbols From DDIFILE. APS reads, extracts, and translates DDIFILE information into DDI symbols, and stores them in *project.group*.DDISYMB.

12 Generate IDCAMS, enter 2, Generate IDCAMS Input Into Amserv in the option field after step 5 completes. The IDCAMS option generates IDCAMS for all files that the subschema references, but more than one subschema can contain a given file. Tailor the IDCAMS so no existing files are deleted. Store the IDCAMS source in your AMSERV data set.

Importing IDMS Database Definitions

Translate database definitions

The IDMS Importer translates IDMS database definitions from your IDMS dictionary into a format usable for generating and precompiling through APS.

Import your IDMS data definitions as follows:

1 Ensure that the IDMS subschema resides in your IDMS dictionary (IDD).

2 Access the IDMS Importer. From the APS Main Menu screen, enter 2 in the Option field. APS displays the Dictionary Services screen. On this screen , enter 1 in the Option field. APS displays the Import Facility screen. On the Import Facility screen, enter 5 in the Option field. APS displays the IDMS Importer screen.

3 Enter the IDMS subschema name in the Member field. You can keep or change the displayed IDMS schema name and version number.

4 If you are importing IDMS12.0 subschemas, ensure that the dataset name of the subchema appears in the IDMS 12.0 SYSIDSM DSN field on the IDMS Options panel.

5 Enter the database name that contains the subschema in the IDMS Dictionary field.

6 Enter 1, Import IDMS Subschema from IDD and Generate DDISYMB, in the Option field.

4 Paint Character Screens

This chapter contains the following sections:

Screen Painter Concepts

Develop screens interactively

The APS Screen Painter lets you paint character-based screens that are intuitive and easy to use. You first paint text, input/output fields, and then blocks of fields that accept multiple record occurrences. You then specify field names, field attribute and edit criteria, and generation parameters. The APS Generators retrieve this screen information from the Application Dictionary to produce native screen source code.

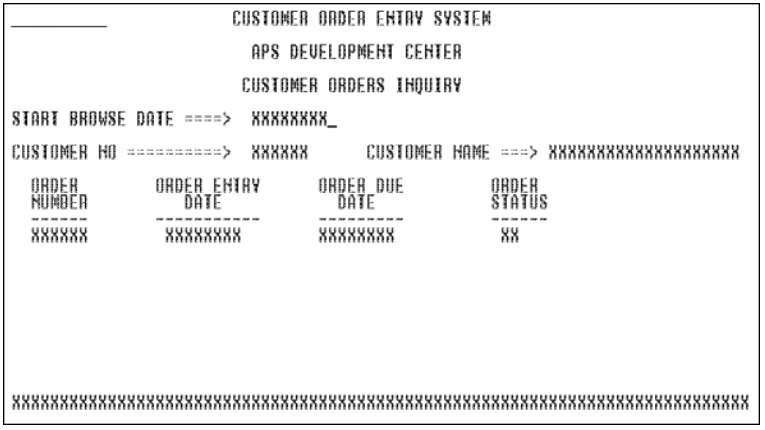
Paint screen fields

A character screen consists of fields and blocks of fields that you paint on a blank screen. You paint two types of fields in the APS Screen Painter:

- Input/Output (I/O) fields that let end users view, add, update, and delete information. You paint I/O fields by typing a string of Xs.
- Text fields that display text, such as prompts for I/O fields, column headings, screen headings, section headings, and explanatory text. You paint text fields by typing any text you want.

Application Screen with Text and I/O Fields shows a sample application screen with I/O fields and text fields.

Figure 4-1. Application Screen with Text and I/O Fields

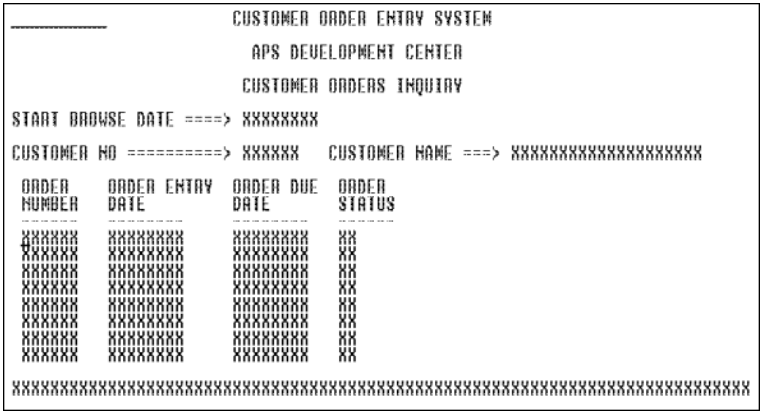


Create repeated record blocks

Your screens can also include repeated record blocks that accept or display multiple occurrences of one or more records. With a simple command, you can repeat a block of one or more source row as many times as necessary. A repeated record block generates a table in Working-Storage.

Sample Application Screen with a Repeated Record Block shows a repeated record block created from the row of I/O fields in *Application Screen with Text and I/O Fields*.

Figure 4-2. Sample Application Screen with a Repeated Record Block



When you create a record block, you do not need to paint and assign characteristics to each field individually--all fields reflect the characteristics of the source row. For example, changing the length of the Order Number field changes the length of all fields in the column.

Choose design options

The APS Screen Painter provides editing and design options to help you paint the screen. For example, you can specify where the Command field automatically appears on your screen, and can determine whether your text displays in upper case, lower case, or both.

Access online help

Additionally, the Screen Painter has an extensive help facility that you can access from your screen by pressing PF1.

Field Attributes

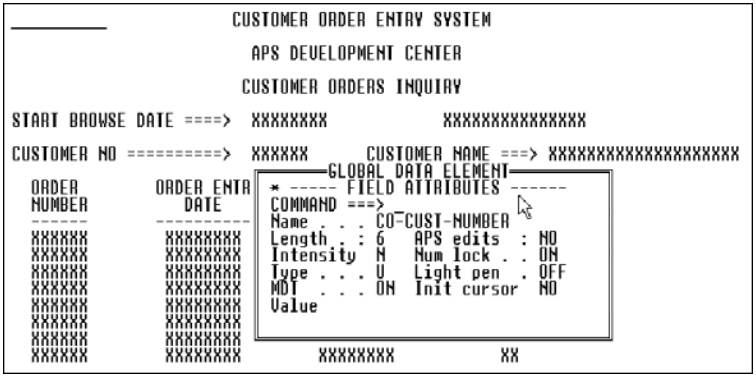
Define 3270 attribute support

APS lets you assign field attributes, such as field protection, brightness, cursor positioning, and color, to both I/O and text fields. The APS Screen Painter supports full and extended 3270 attribute capabilities, including:

- Color
- Underline, blinking, and reverse video features
- Cursor positioning when the screen displays to the end user
- Bright and dark intensity
- Numeric keyboard locking
- Field protection
- Assignment of initial value
- Light pen sensitivity

APS assigns default attribute values to each field for you. Alternatively, you can quickly override the default by entering the values you want, as illustrated in *Field Attributes Screen*.

Figure 4-3. Field Attributes Screen



Field Edits

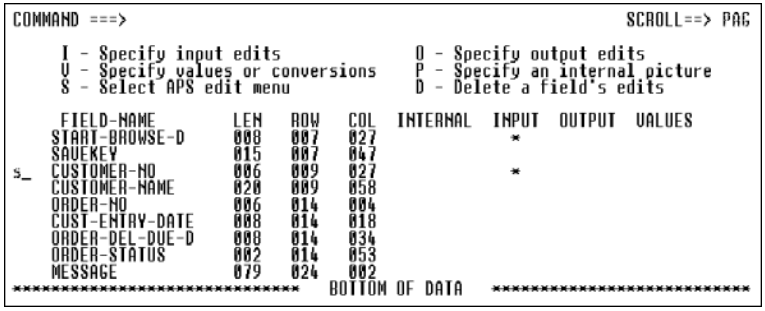
Define the internal, input, and output data representation

Field edits let you define the display and storage characteristics for I/O fields. Field edits can validate input data and format that data for storage and output. You can assign characteristics, such as an internal picture, output picture, edit mask, or date format. Or, you can test for specific values or a range of values.The internal data representation specifies storage characteristics for data in a field. Input and output data representations let you specify the type of data that users can enter or that a field can display. For example, an input data representation for a field may permit a user to enter numbers from 1 to 1,000; an output data representation may require that data display a dollar sign, decimal point, and two places following the decimal point.

You can also code your own edit routines and apply them to multiple screens across any number of application systems. APS field edits ensure that entries match specified definitions. Some fields, however, require specialized testing. For example, if a field has alternate formats, no single field edit can confirm the validity of all possible entries. In such a case, you can write an application edit that verifies all legal entries. Or, you can select a predefined edit from a centralized application edit listing.

Field Selection Screen for Screen Field Editing shows the available edit categories.

Figure 4-4. Field Selection Screen for Screen Field Editing

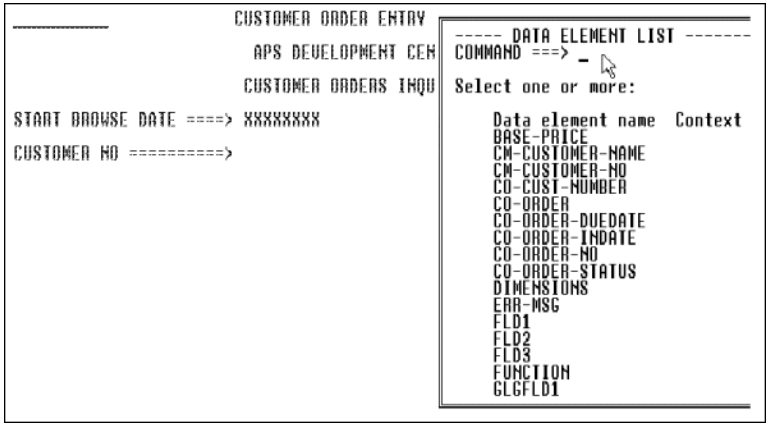


Global Data Elements

Select I/O field definitions

You can select global I/O fields, complete with definitions, attributes, text prompts, and edits, from the APS Data Element Facility. At generation time, APS picks up the definitions in the Data Element Facility.

Figure 4-5. Data Element Facility Listing of I/O Fields



If you modify a global field on your screen, it becomes a local field. APS then stores the field definition as part of your screen member. The local

screen field does not change when the original global field in the Data Element Facility changes.

Know your site and project standards

Depending on how your site or project standard implements the Data Element Facility, you can do some or all of the following:

- Create and modify your own I/O fields.
- Select I/O fields from the Data Element Facility.
- Assign field attributes, assign field edits, or perform other modifications to I/O fields selected from the Data Element Facility.

Before you paint your screens, check with your APS Administrator or Project Leader to determine which of these methods you can use to create and modify I/O fields. The procedures in this chapter cover all methods.

Scenario Prototype

Review screen sequence with users

After you paint several application screens, you can use the APS Scenario Painter to create and run an application model, with or without data. Your end users can view a typical production sequence of screens, enter data into I/O fields, and pass entries between the screens. You do not need to assign field attributes and edits or generate your screens before running screen flow scenario prototypes.

Create the prototyping sequence

To create a scenario prototype, you list the screens in your application in the order you want to view them. For example, if you run the scenario in *Screen Listing in Scenario Painter*, the Customer Order Main Menu displays first, followed by the Customer Record Maintenance screen--just as if an end user requested the screen from the menu. The remaining screens display in sequence and the prototype returns to the menu to exit the application.

Figure 4-6. Screen Listing in Scenario Painter

```
EDIT --- SCENARIO: IDSCEN ------------------------------------ COLUMNS 001 0
COMMAND ===>                                                   SCROLL ===> CSR
-LINE-  -SCREEN- ------------ SCREEN TITLE -------------- -- USER COMMENT --
****** ************************** TOP OF DATA ********************************
000100  IDME     CUSTOMER ORDER MAIN MENU                  MENU SELECTS ONLY
000200  IDCM     CUSTOMER RECORD MAINTENANCE               UPDATE
000300  IDPL     PARTS INVENTORY LIST
000400  IDOM     ORDER RECORD MAINTENANCE
000500  IDOT     ORDER COST TOTALS                         SUMMARY SCREEN
000600  IDOJ     CUSTOMER ORDERS INQUIRY                   BROWSE ONLY
```

Dynamically change prototype at run time

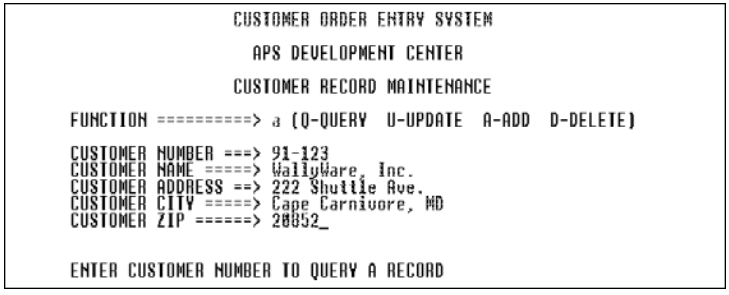
As you run the prototype, you can make changes to correct errors and meet new user requirements by:

- Displaying application screens in any sequence
- Creating and inserting new application screens
- Changing an existing screen
- Entering data in screens

Simulate application data

During the prototype, you can enter data in screen fields and pass the data to other screens. In *Entering Sample Input Data*, the prototype displays the second screen of an 11-screen scenario with user-entered data. When you enter data during a prototype session, you can save it, and reuse it to simulate the movement of data. All data you enter automatically becomes available to other screens that contain identically named fields.

Figure 4-7. Entering Sample Input Data



Target-Specific Parameters

Tailor screen generation

The APS Screen Generator takes your designs from the APS Screen Painter and generates native map definitions. When you are ready to generate, you specify parameters that tailor your screen for the CICS, DDS, IMS DC, ISPF Dialog, or ISPF prototyping target environment.

For any environment, you can:

- Print expanded assembler macros.

- Retain field names as assembler labels.
- Unprotect I/O fields for prototyping.
- Modify I/O field attributes at run time.
- Create a system message field.
- Change how text fields display at run time.

CICS-specific options

For the CICS environment, you can also:

- Generate an assembler END statement.
- Define a unique transaction ID.
- Specify a mapset name.
- Indicate the starting line of the map on the physical screen.

IMS-specific options

For the IMS environment, you can also:

- Generate an assembler END statement.
- Specify standard device characters for different terminals and printers.
- Provide a field for cursor feedback.
- Generate MFS code for logical page requests.
- Define MFS system literals.
- Rearrange the order of input message fields (MID/MOD).
- Assign trancodes, IMS commands, or logical paging commands to PF keys.
- Construct MFS trancodes.

ISPF-specific options

For the ISPF Dialog environment, you can also:

- Generate native definition statements to override ISPF defaults.
- Provide tutorial help panels.
- Control PF key processing.

Painting a Screen

Paint an application screen following the steps below. After step 1, you can perform some or all of the steps in any order.

Access the screen

1 Access the Screen Painter to create or modify a screen in one of the following ways:

- From the Application Painter, enter s in the selection field to the left of the appropriate screen name.
- From the Painter Menu, type e in the Command field, sc in the Type field, and the screen name in the Member field. Then press Enter.

Apply screen design options

2 To specify your editing session options, type profile in the Command field. APS displays a screen displaying the current editing session options in your user profile.

3 Specify editing session options, as follows:

Option	Description	
Command Location	Specify where the Command field appears - enter top for the top-left corner or bottom for the bottom-left corner.	
Caps on/off	*On*	Convert text fields to upper case.
	Off	Preserve or restore text fields as you enter them.
Nulls on/off	*On*	Insert data directly into a row.
	Off	Fill rows with spaces.
Keys on/off	*On*	Display the Screen Painter function key definitions at the bottom of the screen.
	Off	Do not display keys.

Option	Description	
Display field name	*Yes*	Activate the Field Name screen. As you move the cursor between fields by pressing the Enter key, this screen displays the name of the current field. Pressing PF3 removes the screen from display, but keeps this option active.
	No	Do not display the field name. To perform text editing functions, such as typing in a new field or moving fields with the space bar, set this option to No.

4 Press PF3 to set your selections and return to your screen. The selected options remain in effect for all application screens until you change them, either in the current editing session or a subsequent one.

Paint text fields

5 To paint a text field, position the cursor where you want the field to begin and type the text. Text fields can consist of any characters, including special characters. To enter one or more Xs in a text field, you must distinguish the text from an I/O field by putting an underline character on either the left or right side of the X, for example, e_xit.

Paint I/O fields

6 To paint an I/O field, position the cursor where you want the field to begin, and type Xs for the maximum length of the field. I/O fields can have as many characters as can fit on one row of your screen, excluding column 1. Note: You can name the field when assigning field attributes; instructions are later in this procedure.

7 To change the length of an I/O field, move the cursor to the Xs designating the field, and type in your changes. You can space over or delete the Xs, or extend the field with more Xs.

Select predefined I/O fields

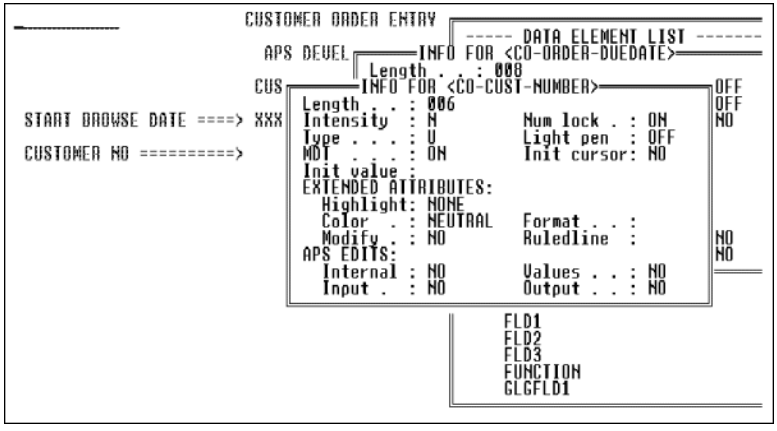
8 To select an I/O field from the Data Element Facility, press PF9 from anywhere on your screen. The Data Element List screen appears on the right side of your screen. An asterisk (*) preceding a name indicates fields that already exist on this screen, and therefore are not available.

9 Navigate the Data Element List screen as follows:

- Enter L xxx in the Command field to locate a portion of the data element list, beginning with field names matching the letters you specify. For example, L ERR redisplays the data element list with the field ERR-MSG as the first entry.
- Press PF7 to scroll backward within the list.
- Press PF8 to scroll forward within the list.
- Press the Tab key or space bar to move to the next element within the list.

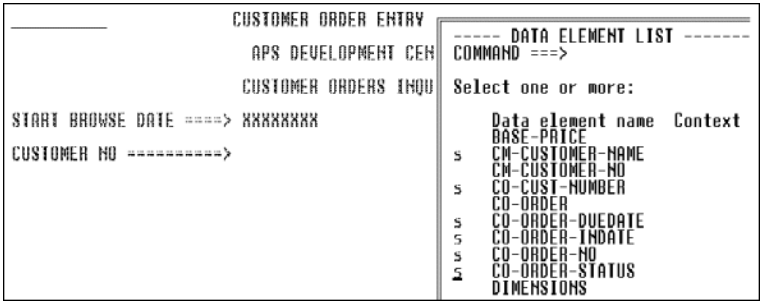
10 To display a field definition, type ? in the selection field preceding the field name(s) you want. The Info screen, as shown in *Data Element Facility Information Screen*, displays the information for the selected screen field. To exit the current Info screen, press either PF3 to take you to the next Info screen if you entered ? on several fields, or PF4 to position you back in the data element list.

Figure 4-8. Data Element Facility Information Screen



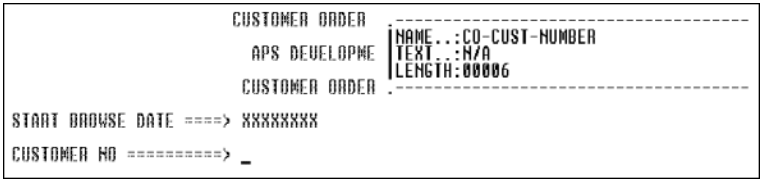
11 To select fields, type s in the selection field preceding the field names you want to include on your screen, and press Enter. A right arrow (>) displays in front of each field you selected. To delete a selection, type d in the selection field, and then press Enter.

Figure 4-9. Selecting Fields from the Data Element List



12 When your selection is complete, press PF3. A screen displays information on the first field you selected from the data element list. The Text field displays the text prompt or label for the field. The Length field displays the field length, excluding any accompanying text from the Text field.

Figure 4-10. Field Selected from the Data Element List



13 Position the cursor where you want to place the field and press Enter. If the field has a text prompt, the prompt begins at the cursor location.

14 The selected field screen presents the fields you selected in alphabetical order. You can use PF keys to manipulate the display of selected fields and the location of the information screen (so that it is not in your way as you position a field), as follows:

PF3 Cancel any selected fields not yet placed on the screen.

PF5 Display first field.

PF6 Display last field.

PF7 Display previous field.

PF8 Display next field.

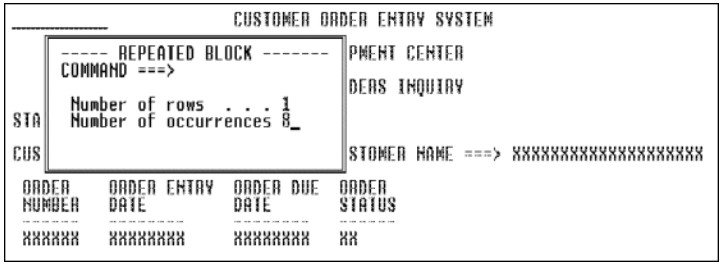
PF10 Move information screen counter-clockwise.

PF11 Move information screen clockwise.

Create repeated record blocks

15 To create a repeated record block from any row(s) of I/O fields you paint, position the cursor on the source row--the top row to be repeated--and press PF7. The Repeated Block screen displays.

Figure 4-11. Creating a Repeated Record Block

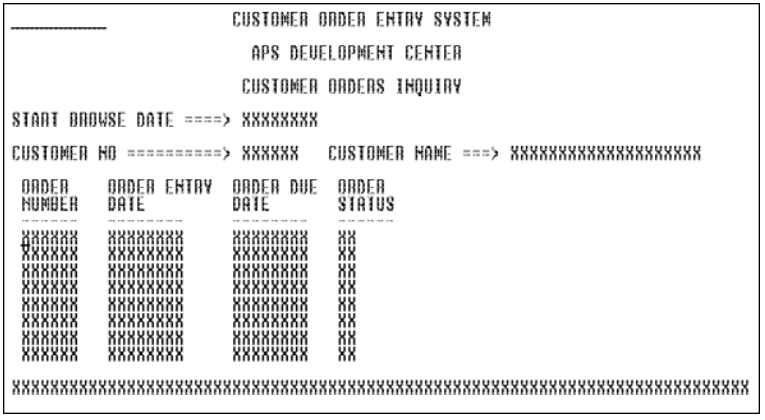


16 Complete the fields in the Repeated Block screen as follows:

- Enter the number of source rows. Each row in a repeated record block encompasses the entire width of the screen.
- Enter the total number of times the block of source rows should occur. For example, a record block that contains two source rows and has five occurrences produces ten lines on the screen. You can create a record block covering as many consecutive blank lines as are available on the screen.

17 Press Enter to create the record block indicated, or press PF3 to exit the screen without creating the record block. *Creating a Repeated Record Block* shows the record block specified in *Repeated Record Block*.

Figure 4-12. Repeated Record Block



Assign a screen title

18 To assign a descriptive screen title that appears when you prototype in the Scenario Painter, enter title or t in the Command field. The Screen Title screen displays. Type your description in the Title field, and press PF3 or enter end on the Screen Title screen Command field. The description does not appear on your screen.

Prototype screen flow for your end user

19 To prototype your screen flow in a scenario prototype to your end user, see *Creating and Running a Screen Flow Prototype*. You do not need to assign field attributes or field edits, or generate the screen, to do a scenario prototype--you can prototype anytime from this point on in the procedure.

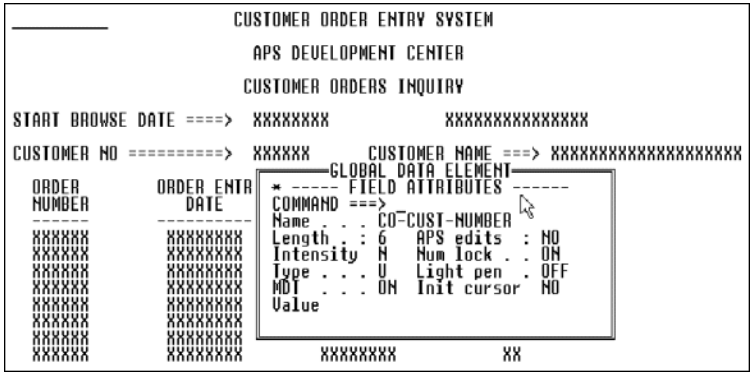
Assign field attributes

20 To assign field attributes by modifying the default attribute values for your text and I/O fields, you can:

- Display attributes for a specific I/O field or text field. To do so, position the cursor on that field and press PF12. Or, to display attributes for the field nearest the current location of the cursor, press PF12. You go to the nearest I/O field, skipping any text fields unless the attributes were previously modified.

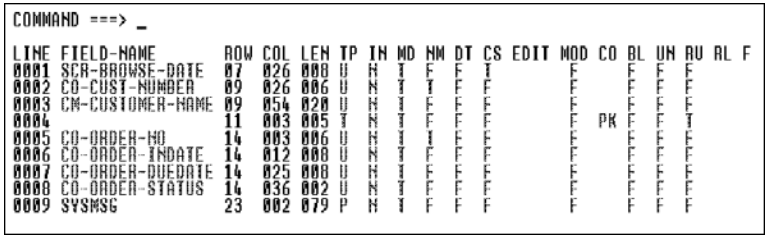
  Field Attributes Screen - Single Field Display illustrates the Field Attributes screen for a single field.

Figure 4-13. Field Attributes Screen--Single Field Display



- Display attributes for all fields. To do so, enter FA in the Command field. A full-screen display of field attributes displays, as illustrated in *Field Attributes Screen - Total Field Display*.

Figure 4-14. Field Attributes Screen--Total Field Display



21 Modify attributes for all screen fields, as follows:

- To modify a field name or attribute, type over the existing value. As soon as you modify the attributes of a field selected from the Data Element Facility, the field becomes a local field and is unaffected by any changes made to the field definition in the Data Element Facility.
- To change attribute values for fields in a repeated record block, modify the applicable fields in the source rows.

- To modify extended attributes for I/O fields, such as color and highlighting, press PF5 from the Field Attributes screen or scroll the full-screen Field Attributes screen to the right.

  Valid attribute values are:

Attribute	Description and Values	
Name	I/O field name; maximum 16 characters. Text fields do not have names because programs do not reference them.	
	Hints: • If you give a screen field the same name as its corresponding database field, APS Online Express automatically maps the relationship for you, prefixing the field name with the screen name; otherwise you must map the screen field to the database in your program. • If the same field appears on several screens, give it the same name on each screen. APS lets you pass data between identically named fields on different screens during scenario prototyping and ISPF prototyping.	
Length	Display field only; to change field length, move the cursor to the Xs designating the field and type in your changes. You can space over or delete the Xs representing the field, or extend the field with more Xs.	
Intensity	B	Bright.
	N	Normal (default).
	D	Dark.
Type	U	Unprotected (default); field is for both input and output.
	P	Protected; field is output only.
	T	Text field with default attributes changed.

Attribute	Description and Values	
MDT	Applies to IMS and CICS only. The modified data tag tells the terminal when to return field data. When this tag is True (T) for a field, the terminal always sends back data; when False (F), the terminal returns data only if the data changes.	
	T	Default for I/O fields; data returned
	F	Default for text fields; returns blanks unless end user modifies the field.
	Note: Set all fields to True when using Field Edits.	
Value	Initial value for screen field; maximum is field length or 27 characters, whichever is less.	
APS edits	Display field indicating if any field edits were assigned to the screen field.	
Num Lock	T	Activate keyboard numeric shift lock
	F	Deactivate numeric shift lock (default)
Light Pen	T	Light pen detectable.
	F	Not light pen detectable (default).
Init cursor	F	Do not position cursor on this field when the program sends the screen. Default for all but the first I/O field.
	T	Position cursor on this field. Default for first I/O field.
Color	BL	Blue
	GN	Green
	NU	Neutral
	PK	Pink
	RD	Red
	TQ	Turquoise
	YL	Yellow

Attribute	Description and Values
Highlight	These are mutually exclusive fields that specify the intended attributes for highlighting a field: BL Blinking UL Underline RV Reverse video Valid values are: T Turns highlighting On F Turns highlighting Off Because the fields are mutually exclusive, you can set only one field to True. The other two fields must be set to False.
Modify	IMS only. F Program cannot modify extended attributes at run time (default). T Program can modify extended attributes. APS generates the extra attribute bytes required.
Format	For KANJI use only. Format field characters for a double-byte character set (DBCS) terminal as follows: Blank Single-byte characters only (default) D Double-byte characters only M Single- and double-byte characters combined
Ruledline	For KANJI use only. Place lines around the field on a DBCS terminal, as follows: spaces No lines L Left side of field R Right side of field O Over field U Under field B Surround field *00-0F* Combination of lines

22 To cycle through and assign attributes to all your I/O and text fields from the Field Attributes screen, press PF12. The screen always appears above or below the active field; the asterisk in the screen points to the active field. Each time you press PF12, the Screen Painter saves the changes made to the current field and moves to the next field. To remove changes for the current field, enter cancel in the Command field.

23 If you assign attributes on the full-screen Field Attributes screen, press PF3 or enter end in the Command field to exit and save your changes, or enter can in the Command field to exit without saving your changes.

Note: For more information on field attributes, see the "Attributes, Screen Fields" topic in the APS Reference.

Create field for system messages

24 To specify the system message field to display both system and program messages, choose one of the following:

- Paint the system message field in any row. Name the field SYSMSG. Assign the Protected field attribute to the field.
- Enter yes or sysmsg in the SYSMSG Message field on the Screen Generation Parameters screen; APS automatically creates the field on the bottom line of your screen.

Paint Online Express program fields

25 If you want the end user to execute program functions by entering a function code on the screen, paint function code fields on your screen. To do so, see *Define Database Access*.

26 For your program to execute database functions, your screen generally needs fields for savekey storage. To paint savekey storage fields, see *Defining Savekey Storage and a Commarea*.

Assign field edits

27 To assign screen field edits, choose one of the following:

- Enter fe in the Command field. The APS Edits Field Selection screen displays, listing all fields defined for your screen. From there you can select a field for edit specification.
- From the Field Attributes screen, press PF4, or enter fe in the Command field. The Edit Selection screen displays for that field. From here you can view, delete, or copy existing edits for your field, or transfer to a specific edit screen.

See *Painting Field Edits* for information.

Assign generation and DC target parameters

28 To assign screen generation parameters for your DC target, enter pm in the Command field. The Screen Generation Parameters screen displays with default parameter values. See *Setting Parameters for Generation* for details.

End the session

29 Choose one of the following to complete your work in the Screen Painter:

- To save your screen and exit, press PF3 or enter end in the Command field.
- To save your screen design and remain in the Screen Painter, enter save in the Command field.
- To exit the Screen Painter without saving your screen design, enter can or cancel in the Command field.

Print screen documentation

30 To print a hardcopy report, see the "About APS Reports" topic in the APS Reference.

Special Considerations

- To create a screen quickly, you may want to copy an existing APS screen and modify it. To do so, use the Create Like function on the APS Painter Menu.
- If APS cannot save your screen, for example, if you don't have enough disk space, a screen lets you specify another data set for storing your screen. Then, before you can access the screen again, you must copy it back to your project.group.APSSCRN file.
- Instead of changing editing options in the profile screen, you can enter commands in the Command field as follows:

Command	Description
bottom	Move Command field to the bottom left corner of the screen.
caps off	Restore text to upper/lower case as entered.
caps on	Convert text to upper case.
keys off	Do not display APS-assigned PF key definitions.
keys on	Display APS-assigned PF key definitions at the bottom of the screen.

Command	Description
nulls off	Fill rows with spaces.
nulls on	Clear rows so you can insert data.
top	Move the Command field to the top left corner of the screen.

- To display a ruler to identify columns, type cols in the Command field, place the cursor where you want the ruler to display, and press Enter. The ruler may overlay painted text. To erase the ruler and replace any overlayed text, enter reset in the Command field.
- APS reserves column 1 for attribute bytes in the generated screen definition.
- I/O fields can have as many characters as can fit on one row of the screen.
- APS allows a maximum of 500 fields per screen. The ISPF prototyping environment allows a maximum of 25 1-byte fields out of the 500 total.
- If you use the screen in an ISPF prototype or your DC target is ISPF Dialog and you need to include an ampersand (&) in a text field:
  - Paint the field as it should appear at run time.
  - Then leave as many spaces at the right of the row as there are ampersands in the row.

  The Screen Generator generates two ampersands for every one you paint. Once the screen is online, ISPF deletes the extra ampersands.
- If you change the field length for a field with assigned field edits, a screen asks you if you want to delete, change, or keep the field edits as they are. Selecting the change option transfers you directly to the Field Edit facility.

Painting Field Edits

Assign screen field edits following the steps below:

Access the field edit facility

1 From your application screen, access the Field Edit Facility in one of the following ways:

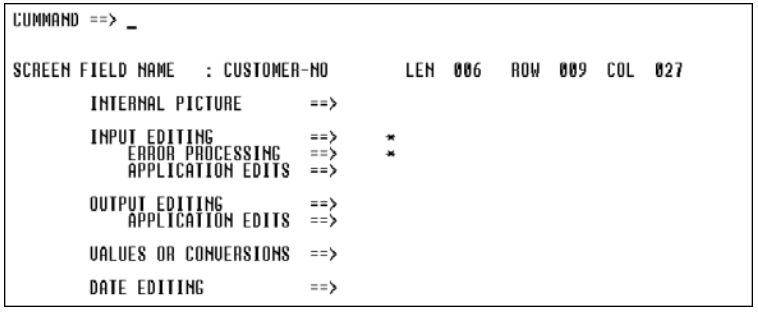
- Enter fe in the Command field. The Field Selection screen displays, listing all fields defined for your screen. An asterisk to the right of a field indicates that edit specifications exist for the field in that category.

Figure 4-15. Field Selection Screen

```
COMMAND ===>                                                  SCROLL==> PAG

     I - Specify input edits                O - Specify output edits
     V - Specify values or conversions      P - Specify an internal picture
     S - Select APS edit menu               D - Delete a field's edits

     FIELD-NAME        LEN   ROW   COL  INTERNAL  INPUT  OUTPUT  VALUES
    START-BROWSE-D     008   007   027              *
    SAVEKEY            015   007   047
s_  CUSTOMER-NO        006   009   027              *
    CUSTOMER-NAME      020   009   058
    ORDER-NO           006   014   004
    CUST-ENTRY-DATE    008   014   018
    ORDER-DEL-DUE-D    008   014   034
    ORDER-STATUS       002   014   053
    MESSAGE            079   024   002
*****************************  BOTTOM OF DATA  ***************************
```

- From the Field Attributes pop-up screen, press F4, or enter fe in the Command field. The Edit Selection screen displays a summary of any edits assigned to that field. An asterisk to the right of an edit name indicates that edit specifications exist.

Figure 4-16. Edit Selection Screen



2 Depending on how you accessed the Field Edit facility in step 1, do one of the following:

- From the Field Selection screen, transfer to an edit specification screen by entering one of the options displayed on the screen to the left of the field name.
- From the Edit Selection screen, enter an s next to the applicable category.

From either screen you can select several fields at one time; they process one after another. When the applicable screen displays, enter values as appropriate; to do so, refer to the topic listed for the category. Select an option as follows:

Option	Description
Edit Selection	Display the Edit Selection screen to see a summary of edits for that field and transfer to other edit specification screens. This is available only from the Field Selection screen.
Internal Picture	Display the Internal Picture screen to specify the internal storage format. See "Related Topics" below later for further information.
Input Editing	Display the Character Input or Numeric Input screen, depending on whether the internal picture specification is character or numeric. The internal picture default type is character. Assign input field edits on these screens. See "Related Topics" below for further information.

Option	Description
Error Processing	Display the Error Processing screen to specify error messages and attributes to display when the data for the field fails input edits. See "Related Topics" below for further information.
Application Edits	Display the Application Editing screen to create your own edit routine to process input data. See the "Application Field Edit Routines" topic in the APS Reference.
Output Editing	Display the Character Input or Numeric Input screen, depending on whether the internal picture specification is character or numeric. The internal picture default type is character. Assign output field edits on these screens. See the "Field Edits" topic in the APS Reference for more information.
Application Edits	Display the Application Editing screen to create your own edit routine to process output data. See the "Application Field Edit Routines" topic in the APS Reference for more information.
Values Or Conversions	Display the Values or Conversion screen to specify a valid value or range of values for input data, or conversion values for either input or output data. See the "Values, Conversion Values, and Value Ranges" topic in the APS Reference for more information.
Special Edits	Display the Special Edits screen to assign date or time specifications. This option is available only from the Edit Selection window. See the "Date and Time Field Edits" topic in the APS Reference for more information.
Input and output editing	Display the Character Input or Numeric Input screen, based on the internal picture specification, followed by the Character Output or Numeric Output screen. To do this, enter io next to a field on the Field Selection screen. See the "Field Edits" topic in the APS Reference for more information.

Copy field edits

3 To copy edits from another field, access the Edit Selection screen for the field you are copying field edits to, and then enter copy in the Command field. The Copy Function screen displays. Enter the field name you are copying edits from; it must be the same length as the current field. The current field inherits the edits of the copied field, and loses any prior edits.

Delete field edits

4 Delete field edits in one of the following ways:

- To delete all field edits for all fields on the screen, access the Field Selection screen, and then enter delete all in the Command field. The Confirm Delete screen displays, where you verify that you want to delete all field edits.
- To delete all field edits for a specific field, access the Edit Selection screen for the field, and enter d after the Internal Picture prompt. The Confirm Delete screen displays, where you verify the deletion.
- To delete a specific field edit for a specific field, access the Edit Selection screen for the field, and enter d to the right of the field edit name. The Confirm Delete screen displays, where you verify the deletion.

Specify global error messages

5 Optionally assign a default error message for the screen to display when the end user enters invalid data as follows:

a Access the Field Selection screen, and enter pm or parm in the Command field. The Parm screen displays.

Figure 4-17. Parm Screen

```
COMMAND ===>

Error message display field ==> MESSAGE           (Required)

   Enter global default message for a field in error below:
> FIELD AT CURSOR IS IN ERROR
   Enter global default message for a required field not entered below:
> FIELD AT CURSOR IS REQUIRED

   Using TP-ATTR attribute syntax, enter any valid attribute combination belo
    ==> POS+BRT

Note: The above error processing occurs only if no error processing
      is specified at the individual field level.
```

b On the Parm screen, type the name of the field that displays the error message.

c Enter the text to display when the data does not pass field edits and enter the text to display when required data is not entered in the appropriate fields.

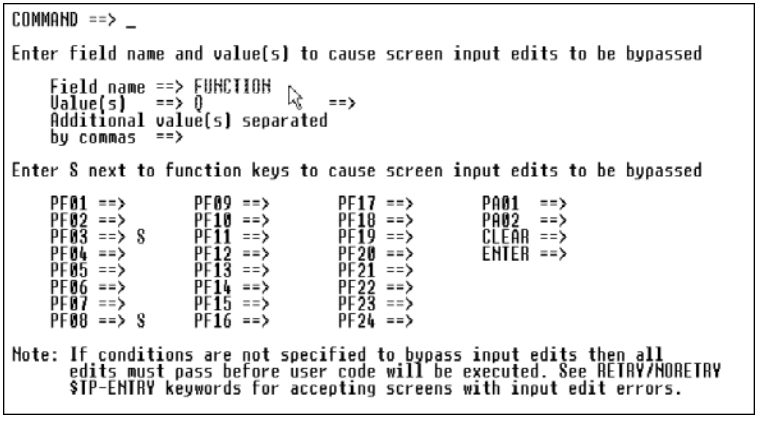
d Specify the attribute values for fields that fail input edits; the default assigns bright and cursor positioning on the field.

You can enter a field-specific error message by selecting the Error Processing prompt on most field edit screens; see the "Error Processing Messages" topic in the APS Reference for information. These messages override the global screen messages assigned in this step.

Specify bypass options

6 To define conditions for bypassing input edits for the screen, press Enter on the Parm screen. A subsequent Parm screen for bypassing edits displays. You can define bypass conditions for one field per screen; if any of these conditions occur, APS bypasses field edits for the entire screen. If the field is in a repeated block, APS bypasses edits for all fields in that row occurrence only.

Figure 4-18. Second Parm Screen



7 Complete the fields on this screen as follows:

Field Name	Specify any field on the screen, including a field in a repeated block, to bypass.
Value(s)	Specify the value or values that let end users bypass input edits. Valid COBOL reserved words are spaces, low-values, and high-values.
Additional Value(s)	Enter as many additional bypass values that can fit on the line; separate each value with a comma.
Program Function Keys	Type s in the selection field to indicate which PF keys the end user can press to bypass the input edits.

Exit the Field Edit facility

8 Choose one of the following to complete your field edits:

- To save your entries and return to the previous screen, press F3, or enter end in the Command field.
- To return to the previous screen without saving any entries, enter can in the Command field.

Creating and Running a Screen Flow Prototype

Test screen sequence

Before you generate your screens, you can review their design and flow with the end user in the APS Scenario Painter. Define a sequence of screens, called a scenario, enter data in those screens, and display the screens to the end user following the steps below. After step 1, you can perform most of the steps in any order.

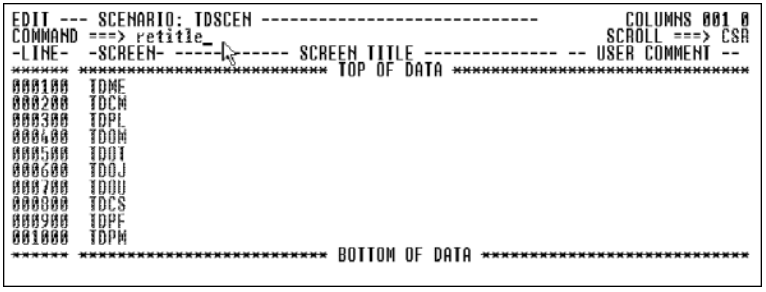
Access Scenario Painter

1 Choose one of the following to access the Scenario Painter:

- To run an existing prototype, access the Application Painter screen for your application and enter run scenarioname in the Command field. The first screen in the prototype displays. Go to step 6 to run the prototype.

- To create, modify, or review a scenario prototype definition:
  - From the Painter Menu, enter CN in the Type field. Then enter the name of the scenario in the Entity field or press Enter to select from a list of scenarios. The prototype definition you specify displays.
  - From the Application Painter, enter CN scenarioname in the Command field. The prototype definition for your application displays. If this is a new prototype definition, the Scenario Painter lists the screens as they appear in your application definition.

Figure 4-19. Initial Application Prototype Definition



Define the prototype

2 To display the titles that you painted in the Screen Painter, enter retitle in the Command field. The titles display in the Screen Title field.

3 To create the prototype definition to represent screen flow, use the ISPF I(nsert), D(elete), C(opy), and M(ove) commands to reorder, insert, and delete screen names until the prototype represents the scenario you want to test. A prototype definition can include up to 160 screens.

4 To describe the screen for the end user, enter text in the User Comments field. For example, a user comment might identify the varying conditions under which the same screen displays. *Initial Application Prototype Definition* shows the definition from *Scenario Prototype Definition*, updated with sequence changes, screen titles, and screen descriptions.

Figure 4-20. Scenario Prototype Definition

```
EDIT --- SCENARIO: TDSCEN ------------------------------------ COLUMNS 001 0
COMMAND ===>                                                   SCROLL ===> CSR
-LINE-  -SCREEN- ------------ SCREEN TITLE -------------- -- USER COMMENT --
****** ************************** TOP OF DATA ********************************
000100  TDME     CUSTOMER ORDER MAIN MENU                  MENU SELECTS ONLY
000200  TDCM     CUSTOMER RECORD MAINTENANCE               UPDATE
000300  TDPL     PARTS INVENTORY LIST
000400  TDOM     ORDER RECORD MAINTENANCE
000500  TDOT     ORDER COST TOTALS                         SUMMARY SCREEN
000600  TDOJ     CUSTOMER ORDERS INQUIRY                   BROWSE ONLY
```

5 To save your prototype definition, enter save in the Command field. To reset the screen flow to its sequence at the beginning of the session, enter reset in the Command field.

Run the prototype

6 To run the prototype, choose one of the following:

- From the Scenario Painter, enter run in the Command field.
- From the Application Painter, enter run scenarioname in the Command field.

The first screen in the scenario definition displays.

7 To display an line that displays scenario information at the bottom of the screen, enter num in the top left corner of the screen. To hide the information line, enter num off.

Figure 4-21. Prototype Information

```
-                    CUSTOMER ORDER ENTRY SYSTEM

                       APS DEVELOPMENT CENTER

                      CUSTOMER ORDERS INQUIRY

START BROWSE DATE ====>  XXXXXXXX            XXXXXXXXXXXXXXX

CUSTOMER NO ==========>  XXXXXX      CUSTOMER NAME ===> XXXXXXXXXXXXXXXXXXXX

  ORDER        ORDER ENTRY      ORDER DUE        ORDER
  NUMBER          DATE            DATE          STATUS
  ------       -----------      ---------        ------
  XXXXXX       XXXXXXXX         XXXXXXXX          XX
  XXXXXX       XXXXXXXX         XXXXXXXX          XX
  XXXXXX       XXXXXXXX         XXXXXXXX          XX
  XXXXXX       XXXXXXXX         XXXXXXXX          XX
  XXXXXX       XXXXXXXX         XXXXXXXX          XX
  XXXXXX       XXXXXXXX         XXXXXXXX          XX
  XXXXXX       XXXXXXXX         XXXXXXXX          XX
  XXXXXX       XXXXXXXX         XXXXXXXX          XX

PF3 = MAIN MENU    PF8 = PAGE FORWARD    PF7 = PAGE BACKWARD
TDOJ     #  6 OF 11 CUSTOMER ORDERS INQUIRY                  BROWSE ONLY
```

In *Prototype Information*, the information line states:

TDOJ	Screen name assigned in Application Painter
6	Sequence number of screen in scenario from prototype definition
11	Total screens in scenario from prototype definition
CUSTOMER ORDERS INQUIRY	Screen title painted in Screen Painter
BROWSE ONLY	User comment entered in Scenario Painter

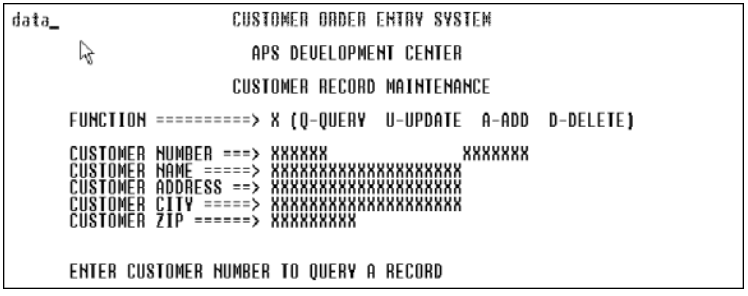
8 To display screens consecutively, press Enter repeatedly until all screens display. If a screen named in your prototype is not yet painted in the Screen Painter, a message displays that information.

9 To transfer to the Screen Painter to create or modify a screen, enter edit in the top left corner of the screen.

Demonstrate data flow between screens

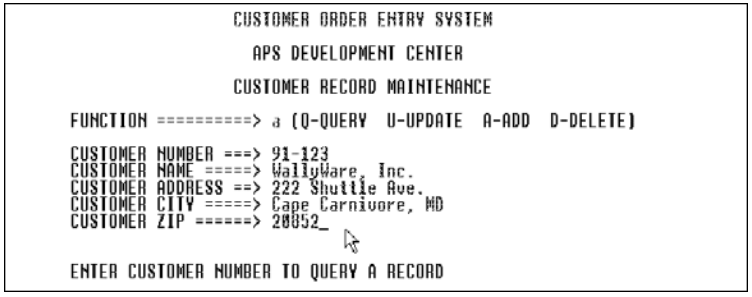
10 To enter data and show data flow between screens, type the data, followed by a space, in the Command area in the upper left corner of the screen.

Figure 4-22. Entering the Data Command



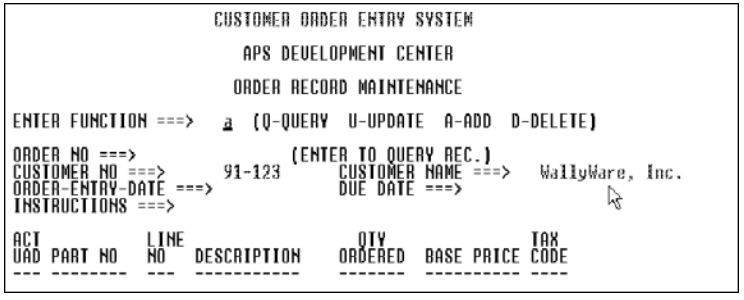
11 Enter data in the I/O fields, as desired. The data you enter replaces the Xs.

Figure 4-23. Sample Input Data



This data automatically appears in other screens that contain identically named fields.

Figure 4-24. Passing Data in the Prototype



12 When simulating data flow in your application, type any command, followed by a space, in the top left corner of the displayed screen:

Command	Description
data	The previous two steps and *Entering the Data Command*, *Sample Input Data*, and *Passing Data in the Prototype* illustrate this option. This option erases the Xs designating I/O fields and activates each field according to attributes assigned in the Screen Painter. You can now enter data in any field; this data automatically displays in identically named fields on other screens.

Command	Description
dataoff	Turn off data simulation and display the screen in its painted format.
read	Display the data saved by the most recently executed SAVE command and execute the DATA command. You can now enter or modify data in any field.
save	Store the current data entered in this scenario for use in future prototyping sessions.

Modify the screen flow sequence

13 To modify the viewing sequence of the screens, type a command, followed by a space, in the top left corner of the displayed screen.

Command	Description
start, first	Display the first screen in the prototype. When you press Enter, the second screen displays, and so on.
last	Display the last screen in the prototype.
end, can, quit	Terminate the prototype and return to the invoking screen.
number	Display the screen in the position specified in the prototype definition.
+increment *-increment*	Display the screen before (+) or after (-) the current screen, according to the prototype definition.
screenname	Display the first occurrence of the specified screen in the prototype definition.
+screenname *-screenname*	Display the first occurrence of the screen specified after (+) or before (-) the current screen.

14 After viewing the last screen in the scenario, press Enter to exit the Scenario Painter.

Modifying Screen Layouts

Once you create a screen, you can easily change its layout. To do so, follow the procedures below.

Delete a Field or Row

Delete screen fields and rows as follows:

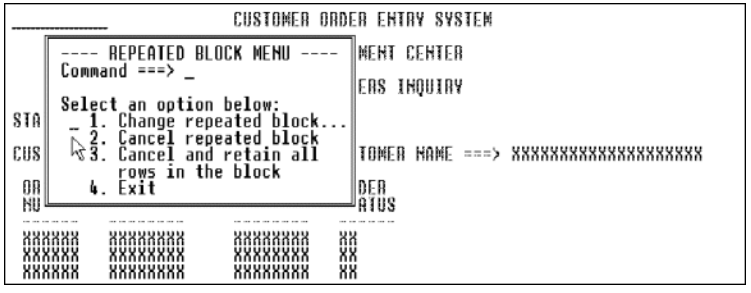
- Write over any field or row using the space bar or other keys.
- Use the Delete key to erase all or part of a field or row.
- Place the cursor anywhere on an I/O field, and press PF6 to delete the field. Press PF6 again to complete the delete.
- Place the cursor on a text field or space in a row, and press PF6 to delete the row. Press PF6 again to complete the delete.

Modify a Repeated Record Block

Modify a repeated record block by following these steps:

1 Position the cursor anywhere within the record block and press PF7. The Repeated Block Menu displays, as shown in *Menu for Modifying a Repeated Block*.

Figure 4-25. Menu for Modifying a Repeated Block



2 Select one of the following options by entering its number in the column preceding the first option listed.

Option	Select ...	To ...
1	Change repeated block	Display the Repeated Block pop-up to change the number of rows or occurrences in the record block.
2	Cancel repeated block	Eliminate the record block and its repeated rows. The original source fields remain.
3	Cancel and retain all rows in the block	Eliminate the record block, but retain every source field. The Screen Painter gives each field a unique name, and thereafter treats each as a separate entity.
4	Exit	Exit the screen without changing the record block. This option is the same as pressing PF3.

Move or Copy a Field or Row

You can move, and if your site standard allows, copy I/O fields, text fields, or entire rows, including those from repeated blocks, to any location on the screen where there is sufficient space. To do so:

1 Position the cursor as follows:

- For an I/O field, place the cursor anywhere on the field and press PF4 to move or PF5 to copy.
- For a text field, mark the boundaries of the text field with the PF10 key. Press PF10 once to mark either the left- or right-most character of the text field; then press PF10 again to mark the opposite side of the field.
- For an entire row, place the cursor anywhere on the row, except on an I/O field, and press PF4 to move or PF5 to copy.
- For a group of fields in a row, position the cursor on the left- or right-most field in the group, as follows:
  - At the first or last character of a text field

- Anywhere within an I/O field

Press PF10. Move the cursor to the opposite side of the group and press PF10 again.

2 Using the arrow keys, position the cursor where you want to move or copy data. APS inserts the row on that line; any previous data on that line shifts down accordingly.

3 Press PF4 to complete the move or PF5 to complete the copy.

4 Alternatively, move a field to another location on the same row by placing the cursor in front of the text or I/O field and either deleting or inserting blank spaces to move the field to the left or right. In this case, you must set the Profile screen field for Nulls to ON.

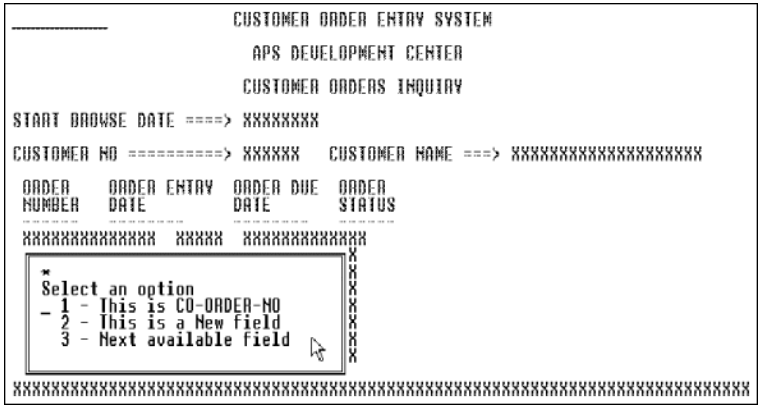
5 Alternatively, move or copy text fields by retyping the entry at a new location and deleting the entry at the old location.

6 To cancel a move or copy at any time, press Enter.

Track Multiple Field Changes

If you add, delete, or modify several fields in the same row at the same time, the APS Screen Painter may prompt you to identify the names of some of the fields resulting from your changes. The Screen Field Name Selection screen displays; the asterisk points to the field in question.

Figure 4-26. Screen Field Name Selection Screen



From this screen, select one of the following options by entering its number in the column preceding the first option listed.

Select ...	To ...
1	Assign the displayed value to the field. In fieldname, the Screen Painter displays an existing field name that cannot be assigned with certainty to a screen field.
2	Let the APS Screen Painter assign a default name to the field. The default name reflects the row and relative position of the field in that row: for example, A-ROW003-FLD002.
3	Default. If more than one existing field name cannot be assigned with certainty to a screen field, you can cycle through those field names by selecting option 3.

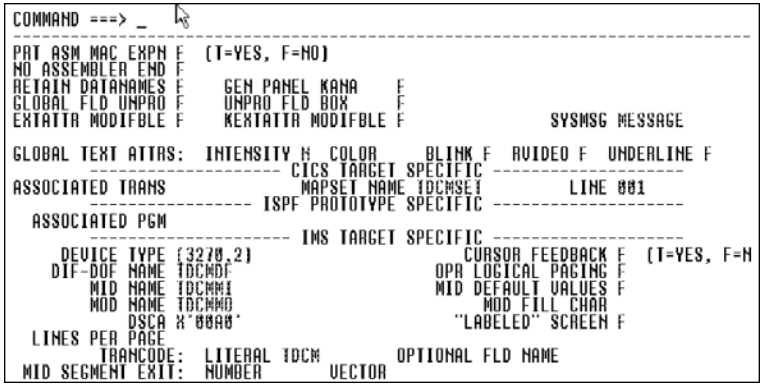
Setting Parameters for Generation

When you are satisfied with your screen designs, you can define the parameters that the APS Screen Generator uses to generate the screens for your data communications (DC) target environment. To do so, follow the steps below. After step 1, you can perform most of the steps in any order.

Access the Screen Generation Parameters screen

1 From the APS Screen Painter, enter pm (parameters) in the Command field. The Screen Generation Parameters screen displays. *Screen Generation Parameters Screen* displays default parameter values for an application screen.

Figure 4-27. Screen Generation Parameters Screen



2 To assign or change any values, move the cursor to the applicable position and type the value. Parameter values of T(rue) and Y(es) are interchangeable, as are F(alse) and N(o).

Assign parameters

3 Change parameter values that affect the screen in any environment, as desired. Applicable parameters and valid values are:

Parameter	Description and Values	
Prt Asm Mac Expn	F	Default. Do not print expanded assembler macros.
	T	Print macros.
No Assembler END	F	Default. Do not generate an assembler END statement.
	T	Generate statement.
Retain Datanames	F	Default. Do not retain painted field names as assembler labels.
	T	Retain field names. Under BMS or MFS, duplicate or invalid names can occur due to the maximum number of characters that BMS and MFS allow.

Parameter	Description and Values	
Exattr Modifble	F	Default. Do not modify extended attributes at run time.
	T	Allow modification at run time; generate EXTATTR=YES and extra attribute bytes in DSECT.
	Anything specified in this field has no effect during prototyping.	
Sysmsg Message	NO or blank	Default. Do not display system messages.
	YES or SYSMSG	Display messages on last line of the screen, if space is available.
	fieldname	Display messages in fieldname.
	YES,row, length|YES, row|YES,, length	Display message of up to length characters on specified row. Row default is last line of screen. Length can be from 40 to 70 characters or up to 131 characters for MOD5 screens.
Intensity	Change the intensity of all text fields.	
	N	Default. Normal
	B	Bright.
Color	Change the color of all text fields.	
	NU	Neutral
	BL	Blue
	PK	Pink
	TQ	Turquoise
	RD	Red
	GN	Green

Parameter	Description and Values
	YL Yellow
Blink Rvideo Underline	Set only one field to T(rue) for text fields. Blinking, reverse video, and underline are mutually exclusive.

Specify CICS parameters

4 For a CICS target, assign parameters as follows:

Parameter	Description and Values
Associated Trans	Specify an associated transaction ID; default is the first four characters of the screen. If more than one screen begins with the same four characters, you need to define a unique transid.
Mapset Name	Override an APS-generated mapset name; maximum seven characters. The default mapset name reflects the number of characters in the screen name, as follows: 4-character name: screennameSET 5-, 6-character name: screenname$ 7-character name: screenname*$*; the $ replaces the seventh character
Line	Starting line of the map on the physical screen; default is 001; value cannot exceed the screen depth.

Specify ISPF prototype parameters

5 For prototyping under ISPF, assign parameters as follows:

Parameter	Description and Values
Global Fld Unpro	F Default. Protect all I/O fields for prototyping. T Unprotect all I/O fields.
Associated Pgm	Name of the program receiving control from the screen; default program name is screenname.

Specify IMS DC parameters

6 For an IMS target, assign parameters as follows:

Parameter	Description and Values
Device Type	Standard device characters for different model terminals and printers. Defaults are IBM-recommended device characters. See your IBM MFS or IMS installation manual for further details.
Cursor Feedback	F Default. Do not define a field in the MID as the cursor feedback field. T Provide cursor information for input processing. To hold the information, APS appends two halfword binary fields to the screen record: screen-CURSOR-ROW and screen-CURSOR-COL. Cursor feedback fields do not affect output cursor positioning.
DIF-DOF Name	Override APS-generated name. Default reflects the number of characters in the screen name, as follows: 4-character name: screennameDF 5-, 6-character name: screenname$ 7-, 8-character name: screenname truncated to 6 characters
Opr Logical Paging	F Default. Do not request operator logical paging. T Request paging. Enter name of field that will contain the paging requests in the Optional Fld Name field.
MID Name	Override APS-generated name. Default reflects the number of characters in the screen name, as follows: 4-character name: screennameMI 5-, 6-, 7-character name: screennameI 8-character name: screennameI; the I replaces the eighth character

Parameter	Description and Values
MID Default Values	F Default. Do not treat initial values as default values for fields in the MFS-generated MID. T Treat initial values as default values.
MOD Name	Override APS-generated name. Default reflects the number of characters in the screen name, as follows: 4-character name: screennameMO 5-, 6-, 7-character name: screennameO 8-character name: screennameO; the O replaces the eighth character
MOD Fill Char	Generate fill characters in the MOD seqment statement. Valid characters are: --, NULL, PT, C, or 'x', where x is any character value.
DSCA	Override the Default System Control Area default value of X'00A0'.
"Labeled" Screen	F Default. Do not append screen name to the input message. T Append the screen name.
Lines Per Page	If device type is a printer, specify number of lines to print on a page.
Trancode: Literal	Specify any literal value as the trancode. Default is the screen name.
Optional Fld Name	Specify fieldname or MFS PFKEY to hold the trancode or operator logical paging command. Alternatively, enter *PF and assign the PF key value on the MFS Function Keys screen, or *TC and construct a trancode on the Trancode Construction screen.
MID Segment Exit: Number Vector	Generate the EXIT parameter on the MID segment statement with Number as the exit routine number and Vector as the exit vector number. Valid values are: Number: 0 to 127 Vector: 0 to 255

7 To save your parameter selections and exit this screen, press PF3. To exit without saving your selections, enter cancel in the Command field.

Note: To learn how to generate an entire application, see *Generate the Application*.

Importing BMS Mapsets

The BMS Mapset Importer creates a screen member from an existing BMS screen description and stores it in APSSCRN.

Acces the importer

To access the BMS Mapset Importer, follow these steps:

1 From the APS Main Menu, select 2, Dictionary Services.

2 Select 1, Importers.

3 Select 4, Screen.

4 Select 1, Import BMS Mapset. The APS Screen Importer panel appears.

Figure 4-28. BMS Importer Screen

5 Define Processing Logic

This chapter contains the following sections:

Concepts of Processing Logic

Define processing logic

You complete your application by defining its processing logic using Online Express, a menu-driven painter that offers a fill-in-the-blanks approach. Online Express references the information that you have specified in the other APS painters and importers, and prompts you to define the processing logic for those specifications. You do the following to complete your application in Online Express:

Eight tasks you can perform with Online Express

- Select predefined program functions. Online Express provides predefined program function logic, including teleprocessing and database read and write functions. You simply select the program function codes that you want.
- Define custom program functions. You can define your own program functions to supplement the predefined functions. End users can execute custom functions just as they execute any predefined function.
- Specify methods for executing functions. You specify the method by which the end user executes the functions. For example, the end user can either enter a code in a function field or press a key.
- Map screen fields to database fields. Online Express automatically displays all screen fields that you have defined in the APS Screen

Painter, so that you can map them to the appropriate fields in your database.

- Define database access. For each database function that you select, you define one or more database calls that specify which record or records to read, and which database actions to perform on them, such as obtain, modify, store, and erase. This task is described in *Define Database Access*.
- Customize the predefined functions. You can modify and supplement the default processing logic of the predefined functions as follows:
  - Add your own logic at predefined locations in your program, called control points. This task is described in both this chapter and in *Define Database Access*.
  - Override the default error processing of database calls. This task is described in *Define Database Access*.
- Define savekey storage. You define savekey storage area(s) to store record key values during program execution if your program must do any of the following:
  - Update records with the U(pdate), A(dd), and D(elete) program functions.
  - Obtain records sequentially with the N(ext) program function
  - Display repeated record blocks that the end user can scroll with the F(orward) and B(ackward) functions.
  - Re-read repeated record blocks so that the end user can update and delete them with the M(odify) and E(rase) functions
- Define Commarea storage. You use a program Commarea to store any data that your program passes between programs with the X(CTL), M(SG-SW), or C(all) functions.

Predefined Program Functions

Online Express provides predefined teleprocessing and database function logic. You simply select the predefined function codes that you need.

Teleprocessing functions

Teleprocessing (TP) functions transfer screen data and program control from the current program to another screen or program. The predefined TP functions include the following:

TP Function	Description
S(end)	Transmits an input/output screen.
M(SG-SW)	Schedules a new program and optionally passes a screen record or other data record to it.
X(CTL)	Transfers control to another program.
C(all)	Calls a subroutine or performs a CICS LINK.
C(lear)	Moves spaces or low-values to all I/O fields.
E(xit)	Terminates the program.

Database functions

Database functions read from and write to your application's database. The predefined database functions include the following:

Read Function	Description
Q(uery)	Obtains one or more records and displays data on the screen.
B(ackward)	Pages backward through a repeated record block.
F(orward)	Pages forward through a repeated record block.
N(ext)	Retrieves the next sequential record and displays data; not applicable to SQL.
R(efresh)	Re-reads the database when the end user executes any database write function on one or more repeated record block rows, and re-displays the record block to reflect the database updates.

Write Function	Description
A(dd)	Stores records.
D(elete)	Erases records.
U(pdate)	Modifies records.

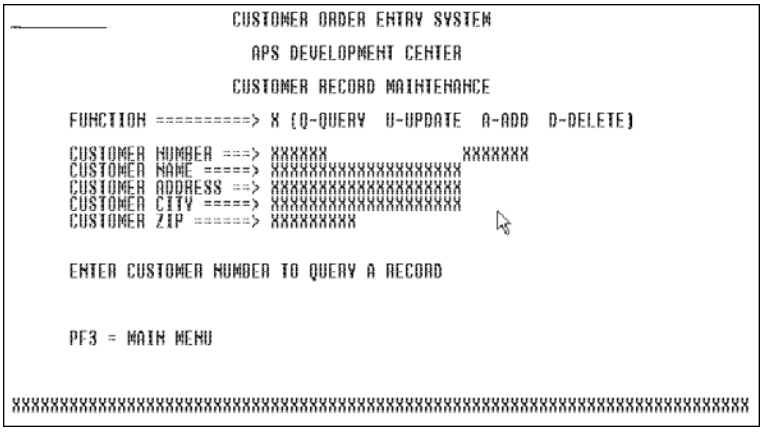
Screen design dictates how you use functions

The screen design dictates which functions act on which fields. When you painted your screen, you designed it to display data in one of the following three formats:

One occurrence of data at a time. For example, your screen might display the name, address, and other information about a particular

customer, as shown in *Screen Displaying One Occurrence of Data*. To query the record in this example, the end user enters q(uery) in the Function field and a value in the record key field, Customer Number. The developer has assigned the S(end) function to the PF3 key so that the end user can press PF3 to display the Main Menu.

Figure 5-1. Screen Displaying One Occurrence of Data

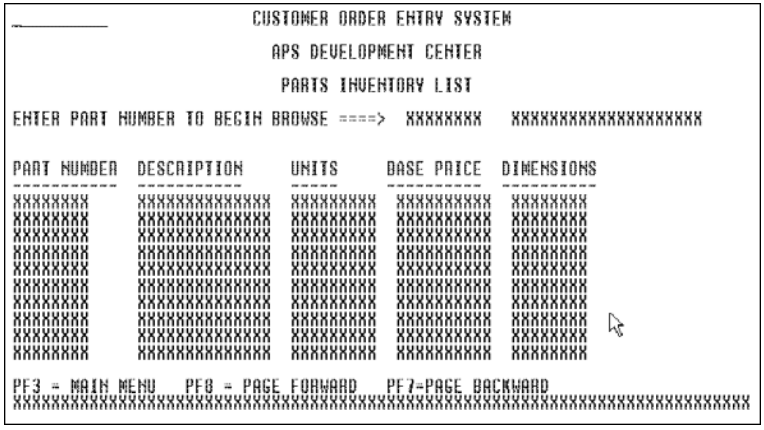


On such a screen, any function that you define for the program acts on all fields. You can select any of the following functions:

A(dd)	*D*(elete)	*N*(ext)	*S*(end)
C(all)	*E*(xit)	*Q*(uery)	*U*(pdate)
C(lear)	*M*(SG-SW)	*R*(efresh)	*X*(CTL)

Multiple occurrences of data, displayed in a repeated record block. For example, your screen might display rows of information about many items in inventory, as shown in *Screen Displaying Multiple Occurrences of Data*. Because this sample program has just one function--query--a function field is not required. The end user starts the query by entering a part number in the record key field. The developer has assigned the F(orward) and B(ackward) functions to the PF8 and PF7 keys so that the end user can press these keys to scroll through the repeated record block.

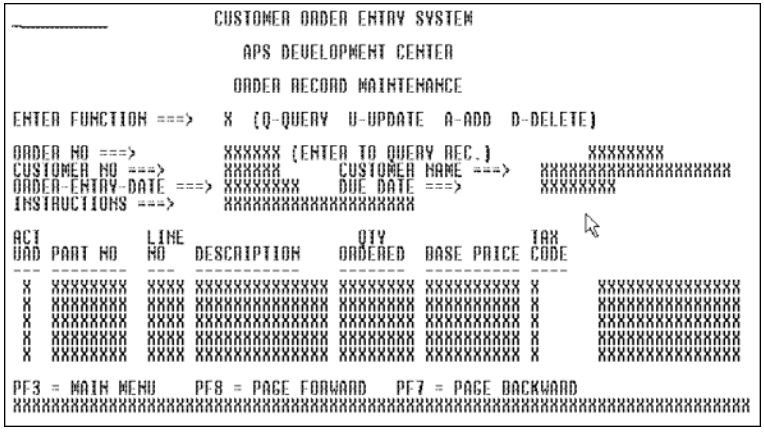
Figure 5-2. Screen Displaying Multiple Occurrences of Data



On such a screen, any function that you define for the program acts on all fields. You can select the same functions that are available for single occurrences of data, plus the F(orward) and B(ackward) functions for scrolling through the repeated block.

Both single and multiple occurrences of data. For example, your screen might display information about a single customer order and a list of the parts ordered, as shown in *Screen Displaying Single and Multiple Occurrences of Data*.

Figure 5-3. Screen Displaying Single and Multiple Occurrences of Data



If your screen displays both single and multiple occurrences of data, you might have to define two function fields:

- A primary function field for updating the single occurrence record
- A row function field, if you must update the repeated record block rows

Use the primary and row function fields to act upon data as follows:

Primary Function Field Functions	**Data Acted On**
Database read functions:	
B(ackward)	All repeated record block row data
F(orward)	All repeated record block row data
N(ext)	All data on the screen
Q(uery)	All data on the screen
R(efresh)	All data on the screen
Database write functions:	
A(dd)	Only the single occurrence data
D(elete)	Only the single occurrence data
U(pdate)	Only the single occurrence data
Teleprocessing functions:	
C(all)	All data on the screen
C(lear)	All data on the screen
E(xit)	All data on the screen
M(SG-SW)	All data on the screen
S(end)	All data on the screen
X(CTL)	All data on the screen
A(dd)	A row of the repeated record block
D(elete)	A row of the repeated record block
U(pdate)	A row of the repeated record block

Screen Displaying Single and Multiple Occurrences of Data illustrates a screen displaying both single and multiple occurrences of data--information about a single customer order and a list of the parts ordered. The end user starts the query by entering q(uery) in the

primary function field and a value in the record key field, Order No. Data displays in all fields. To update, add to, and delete from:

- The customer order information, the end user enters function codes in the primary function field, Enter Function
- The parts records in the repeated block, the end user enters function codes in the row function field, Act.

The developer has assigned the F(orward) and B(ackward) functions to the PF8 and PF7 keys so that the end user can scroll through the repeated block.

Other processing for multiple occurrences of data

You might want to process multiple occurrences of data for purposes other than displaying it in an updateable repeated record block. For example, you might want to:

- Query a single record that has multiple values, such as 12 monthly sub-totals, and insert logic at a control point to move the data to a non-updateable repeated record block.
- Loop on multiple records and display the data in a non-updateable repeated record block.
- Loop on a record and insert logic at a control point to calculate record totals, and display just the totals in one occurrence of data.

Specifying Predefined Program Functions

To specify predefined program functions, follow these steps:

1 Ensure that you have done the following:

   - Listed the components of your application on the Application Painter screen.
   - Painted your program screens using the APS Screen Painter.
   - Generated your program subschema(s) using the APS Database Importers.

2 Display the Application Painter screen.

Access Online Express

3 To start defining program functions for your first program, display the Online Express menu by entering ox in the selection field next to the program name.

Figure 5-4. Online Express Menu

```
COMMAND ===> _

 PROGRAM: TDOJ       SCREEN: TDOJ       SUBSCHEMA: TDDB2      DC TARGET: CICS


   1   PROGRAM DEFINITION  - Specify program information and functions
   2   ALTERNATE FUNCTIONS - Define application and IP functions
   3   PF KEY FUNCTIONS    - Assign PF key functions
   4   FIELD MAPPING       - Map screen fields to program fields
   5   CONTROL POINTS      - Add application specific logic
   6   DATA BASE ACCESS    - Specify data base access
   7   SAVEKEY DEFINITION  - Specify SAVEKEY storage requirements
   SC  APS SCREEN PAINTER  - Invoke APS Screen Painter
   P   EXPRESS PARMS       - Specify Express Parms


COMMANDS:   SAVE  -  COPY <name>  -  GEN  -  CAN  -  AUTO  -  REPORT  -  OXIN
```

4 Display the Program Definition screen by selecting Actions Program Definition, or entering option 1 in the Command field.

Figure 5-5. Program Definition Screen

```
COMMAND ===>

FUNCTION FIELD ===> function              (Field where user enters func code)

FUNCTION CODES ===> quadc
 (Valid Codes: Q=Query U=Update A=Add D=Delete N=Next F=Forward B=Backward
               C=Clear E=Exit)

ROW FUNCTION FIELD ===> row-function      (Field where user enters row code)

ROW FUNCTION CODES ===> uad
 (Valid row codes: U=Update A=Add D=Delete)


SYSMSG FIELD   ===> message_              (Field where messages are displayed)
```

Specify functions and function field(s)

5 Specify any of the following:

- All database read and write functions that you want to include in your program
- A primary function field, if you want the end user to execute the functions by entering codes in a function field

- A row function field, if you want the end user to execute database write functions for repeated record block rows by entering function codes in a function field
- The C(lear) and E(xit) teleprocessing functions; you define the other teleprocessing functions--S(end), X(CTL), M(SG-SW), and C(all)--in step 7
- A field for displaying system messages
- The initial cursor position

To specify the functions and fields above, complete the Program Definition screen fields as follows:

Field	Value
Function Field	The COBOL name of the primary function field where the end user enters function codes to execute program functions. This field is optional if you define just the q(uery) or a(dd) function. If you named this field FUNCTION or FUNCTION-name when you painted your screen, Online Express automatically displays the function field name.

Function Codes	Valid values:
	• Database read functions:
	  • *Q*(uery)
	  • *N*(ext); not applicable for SQL
	  • *F*(orward) and *B*(ackward); applicable only for repeated record blocks
	  • *R*(efresh)
	• Database write functions:
	  • *U*(pdate)
	  • *A*(dd)
	  • *D*(elete)

Field	Value

	• Teleprocessing functions:
	  • *C*(lear). Moves spaces to all I/O fields.
	  • *E*(xit)
	
	Note: You can rename these default function codes with your own codes later in this procedure.

Row Function Field	The COBOL name of the function field where the end user enters database write functions that act only on repeated record block rows.
Row Function Codes	Valid values: • U(pdate) • A(dd) • D(elete)
SYSMSG Field	The COBOL name of the system message field. If you named this field SYSMSG when you painted your screen, Online Express automatically displays this name.
Position Cursor on Field	By default Online Express positions the cursor on the function field for the non-repeated record block data. If you want to override this default with a different field, do one of the following: • To use the initial cursor position field that you specified in the Screen Painter Field Attributes screen, blank out this field with spaces. • Or, specify the override field in this field.

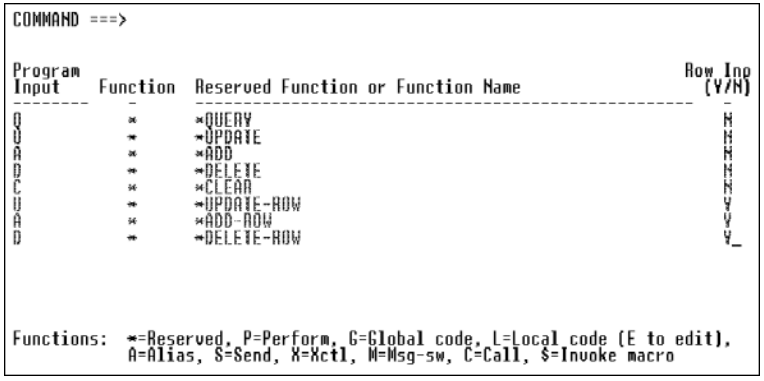
6 Press PF3 to return to the Online Express Menu.

Specify other teleprocessing functions

7 To specify the *S*(end), *X*(CTL), *M*(SG-SW), or *C*(all) teleprocessing functions, select Actions Alternate Functions, or enter option 2 in the Command field. The Alternate Functions screen displays,

showing all function codes that you selected on the Program Definition screen, as illustrated in *Program Definition Screen*.

Figure 5-6. Alternate Functions Screen



8 Complete the screen fields as follows:

- Enter in the Function field any of the predefined teleprocessing function codes that you need--S(end), X(CTL), M(SG-SW), or C(all).
- Enter the objects of the functions - such as the screen to send or the program to transfer to -in the Reserved Function or Function Name field.
- Enter in the Program Input field the function code value that you want the end user to use to execute the function. The value can be 1-8 alphabetic characters.

Notes:

- To rename any default Program Input code with your own alternative alias code, see step 10.
- You use the *P*(erform), *G*(lobal code), *L*(ocal code), and *$* (invoke macro) codes to define custom functions. See *Defining Custom Program Functions*.

Specify how the end user executes the functions

9 Specify how you want the end user to execute the functions by choosing one of the following:

- Default. The end user enters a function code in your screen's function field(s). The codes are those that you specified on the Program Definition and Alternate Functions screens.
  - To rename the default codes with your own codes, perform step 10.
  - If you accept this execution method and the default codes, skip to step 17.

Note: The Enter key is the default function processing key. It causes your program to test the function code that the end user enters, and execute the function. To override the Enter key as the default function processing key, perform step 15.

- The end user presses a function key, such as PF3. To assign one or more functions to a function key, perform step 12.
- The end user presses a special key, such as a PA key. To assign one or more functions to a special key, perform step 14.

Note: You can specify any combination of execution methods. For example, you can assign the E(xit) function to the F3 key, and other functions to function codes.

Rename the default function codes with alias codes

10 To rename any default function code with your own alternative, or alias, function code, go to a new line on the Alternate Functions screen and enter the following values in the following fields:

Program Input	Function	Reserved Function or Function Name
The new code, up to eight characters	*A*(lias)	The function whose code you are renaming, such as **Query* or *s*

11 Press PF3 to return to the Online Express Menu.

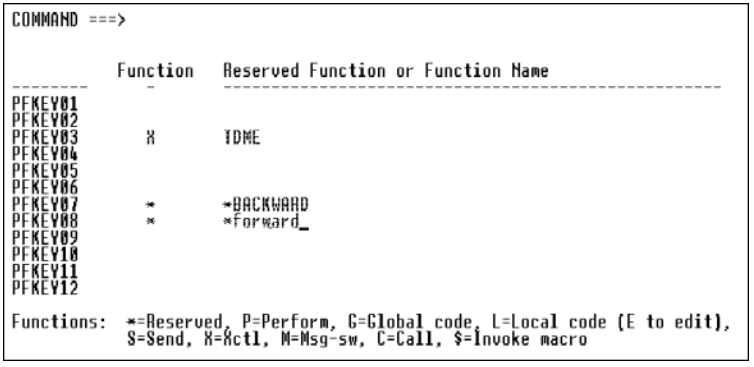
Assign functions to function keys

12 To assign any function to a function key, select option 3, PF Key Functions. The PF Key Functions screen displays, and lets you assign

functions to all 24 function keys. Initially, only the first 12 keys appear on the screen; to assign functions to function keys 13 through 24, select Actions List Next 12 PF Keys, or press Enter.

Note: If you defined trancodes for your MFS mapsets, do not assign functions to function keys.

Figure 5-7. PF Key Functions Screen



13 Press PF3 to return to the Online Express Menu.

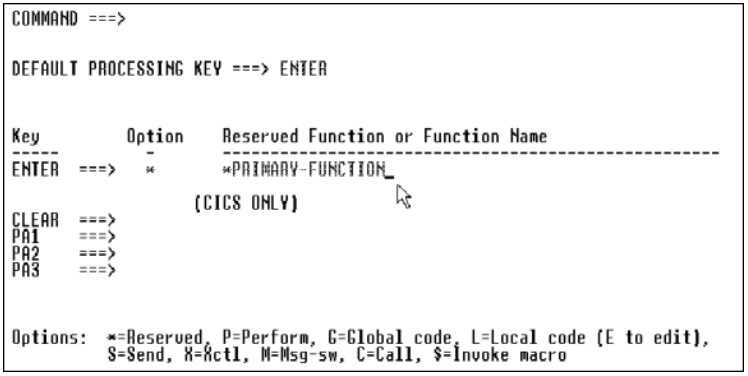
Assign functions to CICS special keys

14 To assign functions to the Clear key and PA keys for a CICS application, select Actions, Special PF Keys, or enter spc in the Command field of the PF Key Functions screen to display the Special Key Definition screen. Complete the screen fields as follows:

a Enter the teleprocessing function codes that you need in the Function field next to a key.

b Enter the objects of the functions, such as the screen to send or the program to transfer to, in the Reserved Function or Function Name field.

Figure 5-8. Special Key Definition Screen



Change the default processing key

15 To override the Enter key as the default function processing key, enter the overriding key name, such as clear or pf10, in the Default Processing Key field on the Special Key Definition screen. The processing key causes your program to test the function code that the end user has entered, and execute the function.

16 Press PF3 to return to the Online Express Menu.

Special Considerations

Clearing screens with low-values

By default, the C(lear) function clears all I/O screen fields with spaces. Alternatively, you can clear repeated block row fields with low-values. To do so, display the Express Parms screen by entering p in the Command field of any Online Express screen, and change value of the Clear With Low-Values parameter to Yes.

Custom Program Functions

Tailor your programs

Without leaving Online Express, you can write custom program functions to supplement the predefined functions provided by Online Express. End users can execute custom functions just as they execute any predefined function.

Write local or global custom function logic

You can write functions specifically for one program, or for use throughout your application. A program-specific custom function is known as a local program stub; a custom function that you use throughout your application is known as a global program stub. Alternatively, you can write a function in a macro and invoke the macro in any program of any application. Stubs and macros are more fully described below:

Custom Function Component	Description
Local stub	Procedure Division and Data Division source that you write and execute specifically in one program. You write a local stub using the Specification Painter, which you access from the Alternate Functions, PF Key Definitions, or Special Key Definitions screen. A local stub can consist of a main paragraph, other paragraphs that the main paragraph performs, and Data Division source code for the paragraphs.
Global stub	Procedure Division source that you can execute in any program of an application. You write a global stub using the Program Painter, which you access from the Application Painter. A global stub can consist of one or more paragraphs.
Macro	Any Customization Facility source that you can execute in any program of any application. You write a macro in the USERMACS data set in your user Project and Group.

Defining Custom Program Functions

To define custom program functions for applications, follow these steps:

Select the function execution method

1 Depending on how you want the end user to execute the function, decide which Online Express screen to use, as listed below:

Execution Method	Online Express Screen
Entering a function execution code	Alternate Functions
Pressing a function key	PF Key Functions
Pressing the Clear key or a PA key	Special Key Definitions

Determine whether to write a stub or macro

2 Determine whether you want to write your function in a local stub, global stub, or macro. You can define any one of them to your program using any of the above screens.

- To define a local stub, perform step 3.
- To define a global stub, perform step 4.
- To define a macro, perform step 5.

Define local stubs

3 To define your custom function in a local stub, follow these steps:

- Depending on how you want the end user to execute the function, display either the Alternate Functions, PF Key Functions, or Special Key Definition screen and complete it as follows:

Alternate Functions	Complete the screen as follows: • Enter a unique function execution code of up to eight characters in the Program Input field. • Leave the Reserved Function or Function Name field blank. • Enter e(dit local code) in the Function field. The Specification Painter screen displays, where you write the stub source code.
PF Key Functions	Complete this screen as follows: • Leave the Reserved Function or Function Name field blank. • Enter e(dit local code) in the Function field next to any function key listed on the screen. The Specification Painter screen displays, where you write the stub source code.

Special Key Definition

Complete this screen as follows:

- Leave the Reserved Function or Function Name field blank.
- Enter *e*(dit local code) in the Option field next to any key listed on the screen. The Specification Painter screen displays, where you write the stub source code.

- Write the local stub as follows:
  - To define the main paragraph, enter your COBOL, COBOL/2, or S-COBOL statements starting in column 12 and continue onto as many lines as you need. Do not enter a paragraph name; APS automatically generates one and displays it at the top of the Specification Painter screen. In the main paragraph, you can perform additional paragraphs that you write in the local stub. For information on writing S-COBOL statements, see the "S-COBOL Structures" topic in the APS Reference .
  - To define an additional paragraph that the main paragraph performs, enter the APS keyword, PARA, in the KYWD column (columns 8-11) and the paragraph name starting in column 12 on the same line. On the following lines, enter your paragraph statements.
  - After all paragraphs, define any Data Division source for the paragraphs, such as data items that the paragraphs reference. To do so, use APS Data Division keywords. For information, see the "Keywords for Program and Specification Painters" topic in the APS Reference .

    For example:

```
-KYWD-  12-*----20---*----30---*----40---*----50---*--
        statement
        .
        .
        .
        perform sub-para-name

 para   sub-para-name
        statement
        .
        .
```

```
        .
ws
01      group-level-data-item
        05  elementary-data-item
        .
        .
        .
```

- When you finish writing the local stub, press PF3 to save it and return to the previous screen. Note that Online Express displays the value L(ocal code) in the Function field. To edit the stub, simply enter e(dit local code) in the Function field.

Define global stubs

4 To define your custom function in a global stub, follow these steps:

- List the global stub name in your application definition. To do so, display the Application Painter screen and enter the following on a separate row anywhere in the definition:
  - In the Programs field, enter the stub name. The name can have a maximum of eight characters. The first character must be alphabetic; others can be alphanumeric or the special characters @, $, or #.
  - In the Screens field, enter the value *stub in the Screens field, to indicate that the stub has no associated screen.
- Write the global stub using the Program Painter as follows:
  - To display the Program Painter, enter s next to the stub name on the Application Painter.
  - To define a paragraph, enter the PARA keyword in the KYWD column and your paragraph name in column 12 on the same line. On the following lines, enter your COBOL, COBOL/2, or S-COBOL paragraph statements. Do not use any other APS keywords in the paragraph. For information on writing S-COBOL statements, see the "S-COBOL Structures" topic in the APS Reference.

    For example:

```
-KYWD-  12-*----20---*----30---*----40---*----50---*--
 para   para-1-name
        statement
        .
        .
        .
```

```
para   para-2-name
       statement
       .
       .
       .
```

- Press PF3 to save your global stub and return to the Application Painter.
- Depending on how you want the end user to execute the function, display either the Alternate Functions, PF Key Functions, or Special Key Definition screen and complete it as follows:

Alternate Functions	Complete this screen as follows: • Enter a unique function execution code of up to eight characters in the Program Input field. • Enter *g* (for global stub) in the Function field. • Enter the stub name in the Reserved Function or Function Name field.
PF Key Functions	Complete this screen as follows: • Enter g (for global stub) in the Function field next to any function key listed on the screen. • Enter the stub name in the Reserved Function or Function Name field.
Special Key Definition	Complete this screen as follows: • Enter g (for global stub) in the Option field next to any key listed on the screen. • Enter the stub name in the Reserved Function or Function Name field.

Define macros **5** To define your custom function in a macro, follow these steps:

- Write your macro in the USERMACS data set in your user Project and Group. See the Customization Facility User's Guide for rules on writing macros.

- List the macro name in your application definition. To do so, display the Application Painter screen and enter the following on any line above your application's program names:
  - In the USERMACS field, enter the name of the USERMACS file that contains the macro. The name can have a maximum of eight characters. The first character must be alphabetic; others can be alphanumeric.
  - In the Loc(ation) field, specify the program location where you plan to invoke the macro. For valid location values, see *Paint the Application Definition*.
- Depending on how you want the end user to execute the function, display either the Alternate Functions, PF Key Functions, or Special Key Definition screen and complete it as follows:

Alternate Functions	Complete this screen as follows: • Enter a unique function execution code of up to eight characters in the Program Input field. • Enter the macro invocation symbol $ in the Function field. • Enter the macro name in the Reserved Function or Function Name field.
PF Key Functions	Complete this screen as follows: • Enter the macro invocation symbol $ in the Function field next to any key. • Enter the macro name in the Reserved Function or Function Name field.
Special Key Definition	Complete this screen as follows: • Enter the macro invocation symbol $ in the Option field next to any key. • Enter the macro name in the Reserved Function or Function Name field.

Mapping Screens to Database Fields

You must map your screen or screen fields to the appropriate fields in your database. To help you do so quickly, Online Express displays all fields that you defined when you painted your screen or screen. You simply specify each screen field's corresponding database field, and indicate whether the screen field is an input field, an output field, or both.

To map screen fields to your database fields, follow these steps:

1 Ensure that you have done the following:

- Listed the components of your application on the Application Painter screen.
- Painted your screens using the APS Screen Painter.
- Imported your program subschema(s) using the APS Database Importers.

Display the Field Mapping screen

2 Select option 4, Field Mapping, from the Online Express Menu to display the Field Mapping screen. The screen displays all screen fields that you defined in the Screen Painter. Note that APS prefixes all field names with their associated screen name.

Figure 5-9. Field Mapping Screen

```
COMMAND ===>

     Screen Field                    I/O/B   Program Field
  -  ------------------------ -----   -     ------------------------------
     TDOM-FUNCTION
     TDOM-ORDER-NO                     B     CO-ORDER-NO
     TDOM-SAVEKEY-1
     TDOM-CUSTOMER-NO                  B     CO-CUST-NUMBER
     TDOM-CUSTOMER-NAME                O     CM-CUSTOMER-NAME
     TDOM-CUST-ENTRY-DATE              B     CO-CUST-ENTRY-DATE
     TDOM-ORDER-DEL-DUE-D              B     CO-ORDER-DEL-DUE-D
     TDOM-ORDER-DEL-INSTR              B     CO-ORDER-DEL-INSTR
     TDOM-ROW-FUNCTION         (AB1)
     TDOM-PART-NO              (AB1)   B     ODL-PART-NO
     TDOM-LINE-NO              (AB1)   B     ODL-LINE-NO
     TDOM-PART-SHORT-DESC      (AB1)   O     PM-PART-SHORT-DESC
     TDOM-QTY-ORDERED          (AB1)   B     ODL-QTY-ORDERED
     TDOM-QTY-BASE-PRICE       (AB1)   B     ODL-QTY-BASE-PRICE
     TDOM-TAX-CATEGORY         (AB1)   B     ODL-TAX-CATEGORY_
     TDOM-SAVEKEY-2            (AB1)

Line cmds:  E=Field edits, I=Insert, D=Delete (Continuation Lines Only)
```

Specify whether fields are input/output

3 Specify whether the screen fields are input, output, or input/output fields by entering i(nput), o(utput), or b(oth) next to each field in the I/O/B column. Leave this column blank for function fields, system message fields, and savekey fields, because they do not have corresponding database fields.

Specify database field names

4 Enter each screen field's corresponding database field name in the Program Field column.

Alternatively, to save yourself some typing, copy all the screen field names to the Program Field column by entering an asterisk (*) in the Command field. Online Express copies all screen field names--except function, system message, and savekey fields--without their prefixes, to the Program Field column. In addition, Online Express enters the value b(oth) in the I/O/B column for all copied fields. Then, modify the names as necessary. To add a prefix to some or all fields simultaneously, use the prefix command.

Enter any of the following prefix command formats in the Command field:

pre fldprefix	Adds fldprefix to fields on all lines
pre fldprefix m n	Adds fldprefix to fields from line m through line n
*pre fldprefix * n*	Adds fldprefix to fields from line 1 through line n
*pre fldprefix n **	Adds fldprefix to fields from line n through the last line

Special Considerations

Qualify fields that belong to multiple records

- If one of your database fields exists in multiple database records, you must qualify the field to indicate which record it belongs to. To do so, insert a line with the i(nsert) line command and on the following line, enter the word of followed by the record name. For example:

```
Screen Field          I/O/B     Program Field
--------------------    -       -------------------
CSINFO-ORDER-NO         B       CO-ORDER-NO
                                  OF SALES-REC
```

Clear the screen

- To clear some or all of the values you entered, enter the reset command in any of the following formats in the Command field:

reset	Clears values on all lines
reset m n	Clears values from line m through line n
*reset * n*	Clears values from line 1 through line n
*reset n **	Clears values from line n through the last line

Control Points

Use control points to tailor default processing logic

Without leaving Online Express, you can write and execute custom processing logic to supplement or override the default logic that Online Express generates. You execute custom logic at any of several APS-provided locations in your program, known as program control points. Control points let you add logic at such locations in the processing logic as:

- Upon program invocation
- Before sending a screen
- Before evaluating program functions
- Before and after moving records between the database and the screen
- Before transferring control to another program
- Before terminating the program normally or abnormally
- Other locations, depending on which functions you define for your program

Control points for database calls

In addition, you can add processing logic before and after database calls. For information, see*Custom Logic at Database Call Control Points*.

Write local or global custom processing logic

You can write local custom logic specifically for one or more control points in a program, or global custom logic for use throughout your application. You execute any local or global logic at any control point.

You write local and global logic in any of the following components in your program or application:

Custom Logic Component	Description
Local stub	Procedure Division and Data Division source that you write and execute specifically in one program. You write a local stub using the Specification Painter, which you access from the Control Points screen. A local stub can consist of a main paragraph, other paragraphs that the main paragraph performs, and Data Division source code for the paragraphs.
Global stub	Procedure Division source that you can execute in any program of an application. You write a global stub using the Program Painter, which you access from the Application Painter. A global stub can consist of one or more paragraphs.
Paragraph	A Procedure Division paragraph and Data Division source that you write specifically for one program and execute at one or more control points. You write a paragraph in the Specification Painter, which you access from the Control Points screen.
Macro	Customization Facility source that you can execute in any program of any application. You write a macro in the USERMACS data set in your user Project and Group.

View contol points on the Control Points screen

The set of control points that might appear in your program is shown below. Because programs vary, you will see a different subset of control points from program to program, depending on which functions you define for them. To view the control points in your program, you display the Control Points screen. In addition, you can look in your generated program source to see where the control points occur; APS generates comments that identify them that you can activate or deactivate. To activate these comments, set the value of the Control Points Comments field to yes on the Express Parms screen. To access the Express Parms screen, enter p in the Command field on the Online Express Menu. The complete set of control points is as follows:

Control Point	Location in Program
After-Receive-Para	After entering a program, regardless of invocation mode.

Control Point	Location in Program
Post-Screen-Read	After a screen-invoked program receives its screen.
Transid-Invoked-Para	After a transid-invoked program is invoked.
Program-Invoked-Para	When APS displays the screen of a program invoked by the XCTL or MSG-SW function.
Pre-Term	Before APS terminates the program.
After-Enter-Check	After the end user presses the processing key (the Enter key is the default), and before the PRE-FUNCTION-TEST paragraph executes.
Pre-Function-Test	Before APS evaluates all functions except the Terminate, or Exit, function.
Pre-Branch	Before each MSG-SW, XCTL, or Call function executes.
Ed-Error-Pre-Send	Before APS send a screen whose field edits have failed.
General-Pre-Send	After APS checks all functions, and before the TP-SEND call executes, when invocation mode is screen-invoked.
Before-Send-Para	Before APS sends the screen, regardless of invocation mode.
Pre-Screen-To-Rec	Before APS performs the MOVE-SCREEN-TO-REC paragraph.
Post-Screen-To-Rec	After APS performs the MOVE-SCREEN-TO-REC paragraph, and the Update or Add function executes.
Pre-Rec-To-Screen	Before APS performs the MOVE-REC-TO-SCREEN paragraph.
Post-Rec-To-Screen	After APS performs the MOVE-REC-TO-SCREEN paragraph, and after the Query function executes.
Pre-RB1-Row-To-Rec	Before the Add or Update function executes for a repeated record block row, and before screen fields move to database fields. APS uses the subscript CTR to reference repeated block rows.

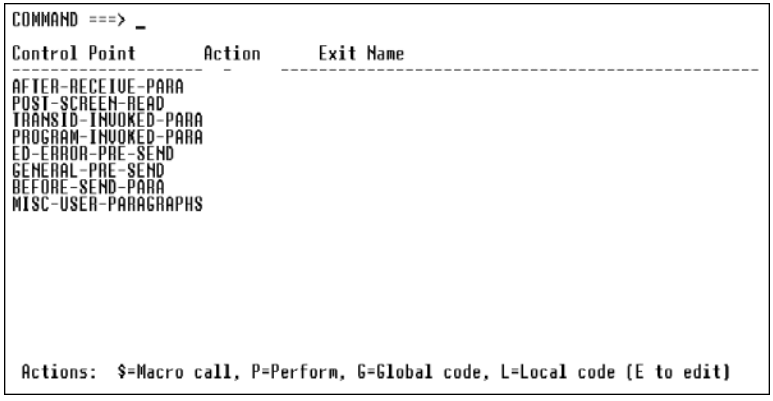
Control Point	Location in Program
Post-RB1-Row-To-Rec	Before the Add or Update function executes for a repeated record block row, and after screen fields move to database fields. APS uses the subscript CTR to reference repeated block rows.
Pre-Rec-To-RB1-Row	After the Query or Forward function executes for a repeated record block row, and before database fields move to screen fields. APS uses the subscript CTR to reference repeated block rows.
Post-Rec-To-RB1-Row	After the Query or Forward function executes for a repeated record block row, and after database fields move to screen fields. APS uses the subscipt CTR to reference repeated block rows.
Error-Send-And-Quit	When a program terminates abnormally, such as when a database call fails when the Database Call Tailoring screen's Abort On Error parameter is set to y.
Misc-User-Paragraphs	A location where you can write and store any number of paragraphs that you can perform at any control point in your program. Write all your paragraphs in one file in this location.

Inserting Logic at Control Points

To insert your custom logic at control points, follow these steps:

1 From the Online Express menu, display the Control Points screen by selecting option 5, Control Points.

Figure 5-10. Control Points Screen



Decide how to implement the control point

2 Determine whether you want to write your custom logic in a local stub, global stub, macro, or paragraph:

- To define a local stub, perform step 3.
- To define a global stub, perform step 4.
- To define a macro, perform step 5.
- To define a paragraph, perform step 6.

Define local stubs

3 To define your control point logic in a local stub, enter e(dit local code) in the Action field next to the control point where you want to execute the stub. The Specification Painter displays, where you write the stub as follows:

a To define the main paragraph, enter your COBOL, COBOL/2, or S-COBOL statements starting in column 12 and continue onto as many lines as you need. Do not enter a paragraph name; APS automatically generates one and displays it at the top of the Specification Painter screen. In the main paragraph, you can perform additional paragraphs that you write in the local stub. For information on writing S-COBOL statements, see the "S-COBOL Structures" topic in the APS Reference.

- To define an additional paragraph that the main paragraph performs, enter the APS keyword, PARA, in the KYWD column (columns 8-11) and the paragraph name starting in

column 12 on the same line. On the following lines, enter your paragraph statements.

- After all paragraphs, you can define Data Division source for the paragraphs, such as data items that the paragraphs reference. To do so, use APS Data Division keywords. For information, see the "Keywords for Program and Specification Painters" topic in the APS Reference.

  For example:

```
-KYWD-  12-*----20---*----30---*----40---*----50---*--
        statement
        .
        .
        .
        perform sub-para-name

 para   sub-para-name
        statement
        .
        .
        .
 ws
 01     group-level-data-item
        05  elementary-data-item
        .
        .
        .
```

b When you finish writing the local stub, press PF3 to save it and return to the previous screen. Note that Online Express displays the value L(ocal code) in the Function field. To edit the stub, simply enter e(dit local code) in the Function field.

Define global stubs

4 To define your control point logic in a global stub, follow these steps:

a List the global stub name in your application definition. To do so, display the Application Painter screen and enter the following on any line above your application's program names:

- In the Programs field, enter the stub name. The name can have a maximum of eight characters. The first character must be alphabetic; others can be alphanumeric or the special characters @, $, or #.

- In the Screens field, enter the value *stub in the Screens field, to indicate that the stub has no associated screen.

b Write the global stub using the Program Painter as follows:

- To display the Program Painter, enter s next to the stub name on the Application Painter screen.
- To define a paragraph, enter the PARA keyword in the KYWD column and your paragraph name in column 12 on the same line. On the following lines, enter your COBOL, COBOL/2, or S-COBOL paragraph statements starting in columns 12. Do not use any other APS keywords in the paragraph. For information on writing S-COBOL statements, see the "S-COBOL Structures" topic in the APS Reference.

  For example:

```
-KYWD-  12-*----20---*----30---*----40---*----50---*--
 para   para-1-name
        statement
        .
        .
        .
 para   para-2-name
        statement
        .
        .
        .
```

c Press PF3 to save your global stub and return to the Application Painter.

d Execute the stub at any control point by entering g(lobal code) in the Action field next to the control point, and the stub name in the Exit Name field.

Define macros **5** To define your control point logic in a macro, follow these steps:

a Write your macro in the USERMACS data set in your user Project and Group. See the ***Customization Facility User's Guide*** for rules on writing macros.

b List the macro name in your application definition. To do so, display the Application Painter screen and enter the following on any line above your application's program names:

- In the USERMACS field, enter the name of the USERMACS file that contains the macro. The name can have a maximum of eight characters. The first character must be alphabetic; others can be alphanumeric.
- In the Loc(ation) field, specify the program location where you plan to invoke the macro. For valid location values, see *Paint the Application Definition*.

c Invoke the macro at any control point by entering the macro invocation symbol $ in the Action field next to the control point, and the macro file name in the Exit Name field.

Define paragraphs

6 To define your control point logic in one or more paragraphs, follow these steps:

a Write the control point paragraph(s) using the Specification Painter. To do so, enter e (for edit) next to the Misc-User-Paragraphs control point on the Control Points screen and write the paragraphs according to the following rules:

- For each paragraph, enter the PARA keyword in the KYWD column and your paragraph name in column 12 on the same line. On the following lines, enter your COBOL, COBOL/2, or S-COBOL paragraph statements starting in column 12. Do not use any other APS keywords in the paragraph. For information on writing S-COBOL statements, see the "S-COBOL Structures" topic in the APS Reference.
- After all paragraphs, you can define Data Division source for the paragraphs, such as data items that the paragraphs reference. To do so, use APS Data Division keywords. For information, see the "Keywords for Program and Specification Painters" topic in the APS Reference.

  For example:

```
-KYWD-  12-*----20---*----30---*----40---*----50---*--
 para   para-name
        statement
        .
        .
        .
```

```
        perform sub-para-name

para    sub-para-name
        statement
        .
        .
        .
ws
01      group-level-data-item
        05  elementary-data-item
        .
        .
        .
```

b When you finish writing the paragraph(s), press PF3 to save it and return to the previous screen. Note that Online Express displays the value L(ocal code) in the Action column. To edit the paragraphs, simply enter e(dit local code) in the Action column.

c Perform the paragraph at any control point by entering p in the Action column next to the control point, and the paragraph name in the Exit Name column. To pass arguments to the paragraph, code them next to the paragraph name without parentheses. For example:

```
paraname arg1 arg2 arg3
```

For more information, see the "TP-PERFORM" topic in the APS Reference.

6 Define Database Access

This chapter contains the following sections:

Concepts of APS Database Access

Database functions and actions

An Online Express database call defines which record or records to read, and which database actions to perform on them. Each database function that you specify in your program definition has a corresponding database action that defines the function, as shown below:

Database Function	Corresponding Database Action
Query	Obtain
Update	Modify
Add	Store
Delete	Erase

For example, if your program must query, update, add, and delete a record, you must define a call to obtain, modify, store, and erase that record. To do so, you simply enter the action codes o(btain), m(odify), s(tore), and e(rase) next to the record that Online Express displays.

Looping on records

You can obtain multiple occurrences of a record simultaneously by looping on the record with the l(oop) action code. For example, you might want to loop on a record to display multiple data items in a repeated record block, list box, or combination box on your screen. Alternatively, you might want to loop on a record to calculate field totals, and display just the calculation results. You can map to your program screen any fields of any records that you loop. You specify which fields to map, using the Field Mapping screen.

By default, Online Express considers any call or calls that follow a loop call to be nested within the loop. That is, these calls execute each time that the loop executes. To indicate that a call is nested within a loop, Online Express assigns a nesting level value to the nested call. You can, however, decrease the nesting level of any call to execute it independently of the loop.

Define inner, or nested, loops

When you define multiple loops in your program, Online Express considers the first loop to be an outer loop, and each subsequent loop to be an inner loop, nested within the previous loop. By default, loops are progressively nested--that is, the second loop is nested within the first, and the third loop is nested within the second. As with any nested call, you can decrease the nesting level of a nested loop to execute it independently of a loop, or nest it within a different loop.

Define database calls

Define a database call by completing a few Online Express screens that prompt you to do the following:

- Select which record or records to access.
- Specify the database read and write actions that you want to perform on the record.
- Qualify the data that you access by specifying field or column criteria.

Online Express displays a list of all records of the program subschema or PSB. From that list, you select a record and specify its read and write actions. For example, to define a call for a record that you want end users to modify and store, simply specify the obtain, modify, and store action codes next to that record in the list.

How you use the action codes in Online Express depends on the structure and methods inherent to your database target. For example, you can obtain data from multiple IMS records in a path and specify that end users can modify and store it. Or you can select multiple SQL

tables and loop on them, returning multiple row records that can be modified, stored, and erased.

Online Express then displays all fields or columns of the record or records that you select for the call. You specify any field or column criteria to qualify the data that the call returns. Online Express prompts you for information appropriate to your selected database target.

Qualify Online Express database calls using methods available to your database target, as shown below:

Database	Qualification Method
IMS	Qualify on any field, including: • Key field • Non-unique search field • Sequence field Qualify on multiple fields and conditions using Boolean operators
SQL	Qualify on any column Qualify on multiple columns and conditions using Boolean operators Qualify on multiple columns of multiple tables, using Union and Join calls Qualify with Subselect specifications, including SQL column and scalar functions, and Exists, Group By, and Having clauses
VSAM	Qualify on any field, including: • Primary index • Partial key field
IDMS	Qualify on any field, including: • Address • CALC key • Key • Non-unique search • Sequence

You should define the calls in the order in which you want to execute them, but you can rearrange the order and modify any call definition at any time.

Execution methods for database calls

Typically, you define calls that execute when the end user enters a function code, presses a key. For example, you might want the obtain action to execute when the end user enters Q, presses F5, or presses the Enter key.

In addition, you can define calls that execute as a custom program function. For example, you can automatically execute a call at various locations in your program, known as control points. Online Express provides many control points at which you can execute calls.

Error processing

Online Express generates status flags that you can use to determine execution errors. Each flag has a default status code, as shown below:

Status Flag	Default Status Code
OK-ON-REC	N(ormal)
END-ON-REC	N(ormal)
NTF-ON-REC	E(rror)
DUP-ON-REC	E(rror)
VIO-ON-REC	E(rror)

Customize database call processing

While Online Express lets you completely define database calls without having to code them, you can also extend and customize those calls to suit your needs. Without leaving Online Express, you can write and execute custom database call processing logic to supplement or override the default logic that Online Express generates. You execute custom logic at any of several APS-provided locations in your program, known as database call control points. The control points let you add processing logic before and after a database call, and when calls execute normally or abnormally.

If you want to override APS error processing routines, you change a status flag's status code from Error to Exception, and then write your own error routines at control points. You can also override the default error messages with your own messages.

Define a Commarea

You can define an area in your program to store any data that your program passes between programs, called a Commarea. You must define a Commarea if your program passes data with the X(CTL), M(SG-

SW), or C(all) functions. You do so simply by specifying its size to Online Express.

Define a savekey storage area

You can also define an area in your program to store record key values during program execution, called savekey storage. You must define savekey storage if you program must do any of the following:

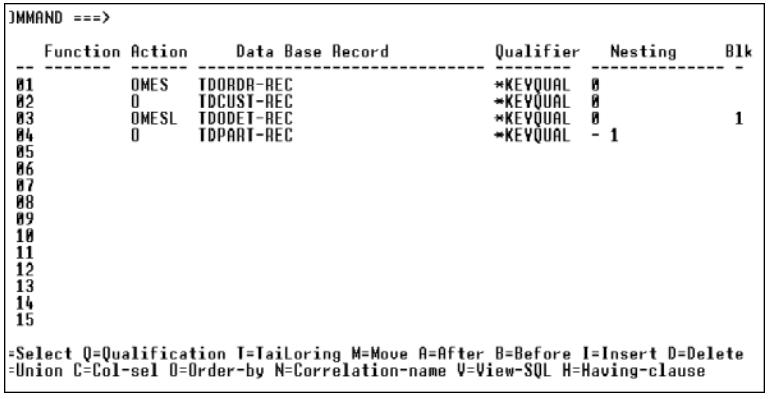
- Update records with the U(pdate), A(dd), and D(elete) program functions.
- Obtain records sequentially with the N(ext) program function.
- Display repeated record blocks that the end user can scroll with the F(orward) and B(ackward) functions.
- Re-read repeated record blocks so that the end user can update and delete them with the M(odify) and E(rase) functions.

You can store savekey data either in savekey screen fields that you define in your screen definition, or in the program Commarea.

Sample database calls

The Database Access Summary screen provides access to all the screens that you need to define a call, and displays a summary list of all calls that you define for a program. *Sample Database Access Summary Screen* illustrates four sample SQL database calls.

Figure 6-1. Sample Database Access Summary Screen



```
)MMAND ===>

   Function Action     Data Base Record           Qualifier   Nesting      Blk
-- -------- ------ ------------------------------ --------  ---------------  -
01          OMES   TDORDR-REC                     *KEYQUAL  0
02          O      TDCUST-REC                     *KEYQUAL  0
03          OMESL  TDODET-REC                     *KEYQUAL  0                1
04          O      TDPART-REC                     *KEYQUAL  - 1
05
06
07
08
09
10
11
12
13
14
15

=Select Q=Qualification T=TailOring M=Move A=After B=Before I=Insert D=Delete
=Union C=Col-sel O=Order-by N=Correlation-name V=View-SQL H=Having-clause
```

Note the following about the calls defined above:

- Call 01 obtains a customer order table record qualified on key field criteria, as indicated by the Qualifier field value KEYQUAL. The qualification is satisfied when the end user enters a screen field value that matches that criteria. The end user can query, update, delete from, and add to the table, as indicated by the o(btain), m(odify), e(rase), and s(tore) action codes in the Action field.
- Call 02 obtains a customer information table record qualified on a field in the customer order table.
- Call 03 loops on the detail table of the customer order table, qualified on the customer order table key field. The end user can also update, delete from, and add to the table, as indicated by the m(odify), e(rase), and s(tore) action codes in the Action field. The records returned by the loop are mapped to a repeated block on the program screen, as indicated by the value 1 in the Blk (Block) field.
- Call 04 obtains a part master table record qualified by a field in the detail table. Because call 04 follows a loop, Online Express assumes that the call is nested within the loop, as indicated by its Nesting field value - 1.

Defining SQL Database Calls

You can define SQL database calls to obtain, loop, modify, erase, and store columns from one or more tables. Specifically, you can define the following types of SQL calls:

Types of SQL calls

- Basic SQL calls, to access qualified columns of one table.
- Join calls and Union calls, to access qualified columns of multiple tables.

Types of call qualifiers

Online Express lets you specify any of the following SQL call qualifiers:

- Select and Subselect statements
- Boolean qualifiers
- SQL column and scalar functions
- Exists and Not Exists clauses

- Group By and Having clauses

Procedures for defining basic calls, Join calls, and Union calls follow, in separate sections.

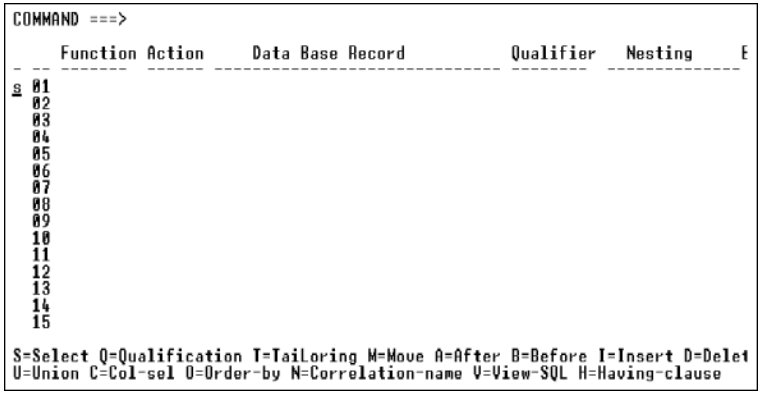
Defining Basic SQL Calls

Follow these steps to define a basic, qualified SQL database call. Procedures for defining Join and Union calls appear in separate topics, later in this section.

Select the Database Access Summary screen

1 Select option 6, Database Access, from the Online Express menu. Alternatively, enter 6 or dba on any primary Online Express screen. The Database Access Summary screen displays.

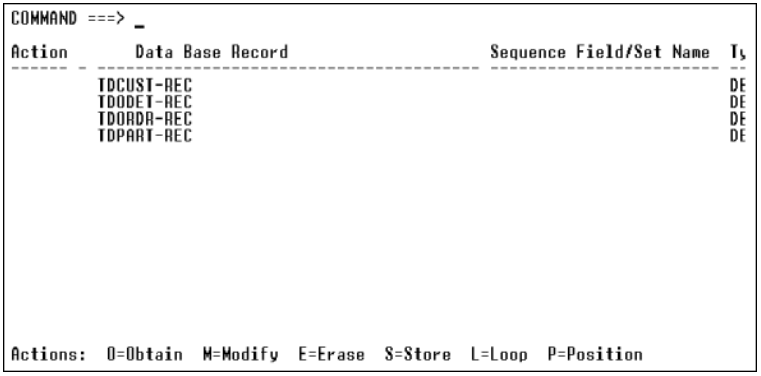
Figure 6-2. Database Access Summary Screen



Define the first call

2 To define the first call, enter s in the selection field next to call number 01. The Database Record Selection screen displays, listing all tables in the program subschema.

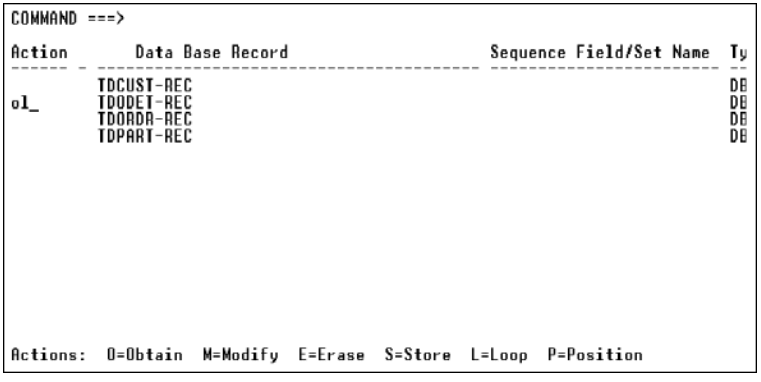
Figure 6-3. Database Record Selection Screen



Specify action codes

3 Enter one or more action codes, in any order, in the Action field next to the table that you want to access in the first call. For example, *Specifying Database Action Codes* illustrates a simple call that obtains and loops on table TDODET-REC, as indicated by the o(btain) and l(oop) action codes next to the table. To allow the end user to modify, erase from, and store to the table, enter the m, e, and s action codes as well. To define nested loops, see *Nested Loops*.

Figure 6-4. Specifying Database Action Codes

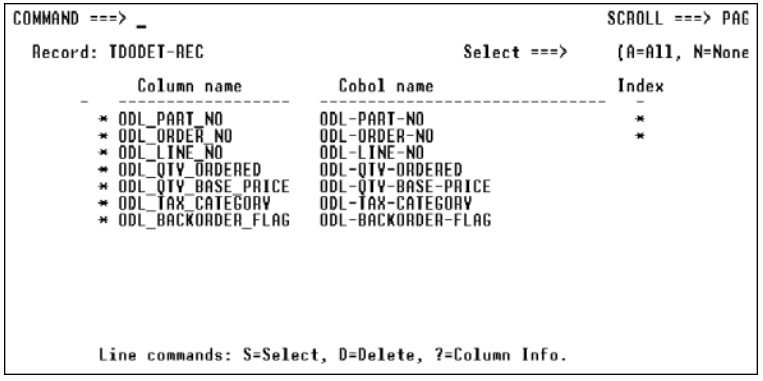


Select columns

4 Press PF3 to display the Column Selection screen, which lists all columns of the selected table, as shown in *Column Selection Screen*.

Note that Online Express identifies each index column with an asterisk in the Index field.

Figure 6-5. Column Selection Screen

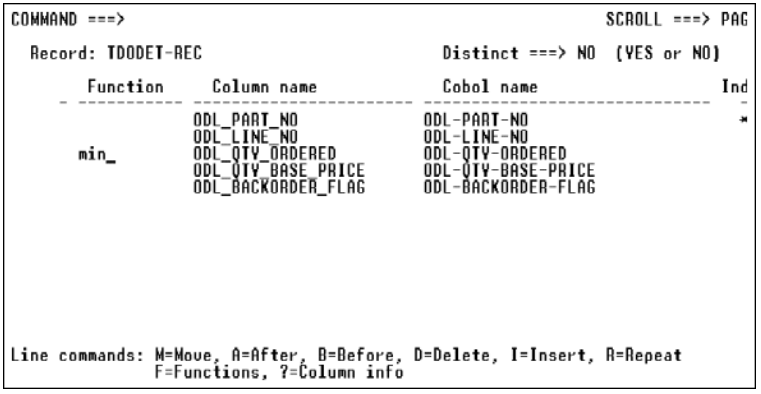


5 Select the columns for the call by entering s in the selection field next to each column that you want to include. Alternatively, enter d(elete) next to each column that you want to exclude. To select all columns, enter a(ll) in the Select field at the top of the screen. To display the definition of any column, enter ? in the selection field next to it.

Update the column list

6 Press PF3 to display the Column Selection Update screen, where you see the list of columns that you selected for the call, as shown in *Column Selection Update Screen*.

Figure 6-6. Column Selection Update Screen



Add, exclude, and rearrange the order of columns

7 Do any of the following:

- To add a column, insert a line with the i(nsert) line command and enter the column name in the Column Name field. Alternatively, insert a line, enter listcol in the Command field to display a column list, and select a column from the list. Then press PF3 to return to the Column Selection Update screen.
- To exclude a column, enter d(elete) in the selection field next to it.
- To rearrange the order of the columns, use the line commands m(ove), a(fter), and b(efore).

Override COBOL host variables

- To override the default COBOL host variables with Working-Storage fields or literals, simply overtype the host variable names in the COBOL Name field. Doing so enables you to change the destination for columns that the call obtains and to update columns with literal values. These changes affect the obtain, modify, and store actions as follows:

Override Value	Effect
Working-Storage field	Obtain action: Obtains the column value into the Working-Storage field. Modify and Store actions: Updates the column with the Working-Storage field value.

Override Value	Effect
Literal	Obtain action: Obtains the column value into the default COBOL host variable. Modify and Store actions: Updates the column with the literal value.

Obtain literal values from a column

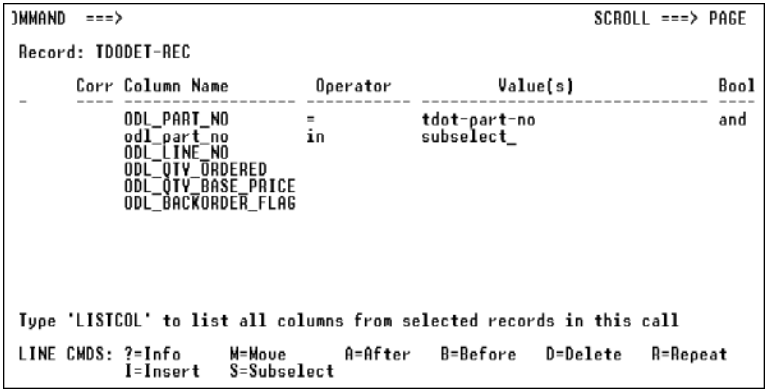
- To obtain a literal value from a column, overtype a column name with the literal, or insert a new line and enter the literal.

Assign SQL functions

- To assign a SQL column or scalar function to one or more columns, enter the function name in the Function field next to the column(s), as shown in *Column Selection Update Screen*. Alternatively, enter f(unction) in the selection field next to the column to display a function list, select one from it, and press PF3 to return to the Column Selection Update screen. When you assign a column function--AVG, COUNT, MAX, MIN, or SUM--to at least one, but not all, columns of a loop call, Online Express generates a Group By clause. The Group By clause lists all other columns, called grouping columns, in the order in which they appear on the Column Selection Update screen. You can rearrange the order using the line commands m(ove), a(fter), and b(efore). To define an optional Having clause to specify conditions that the group must satisfy, see step 11.

8 Press PF3 to display the SQL Qualification Specification screen, where you qualify the columns that you selected, as shown in *SQL Qualification Specification Screen*.

Figure 6-7. SQL Qualification Specification Screen



Qualify columns

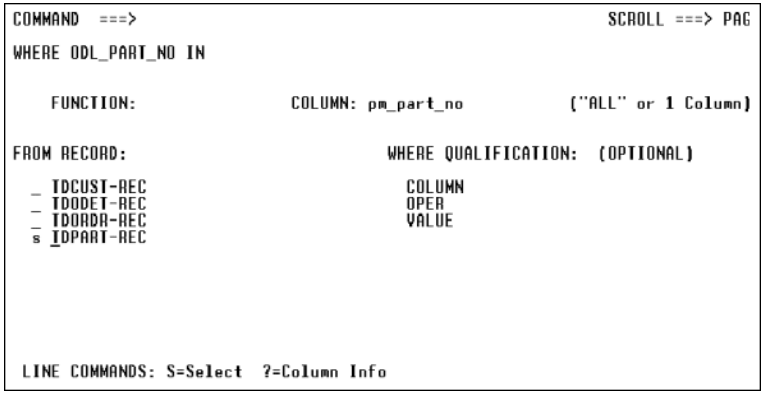
9 Qualify one or more columns by entering an operator and a qualification value next to the column(s). A qualification value can be a:

- COBOL host variable; APS automatically generates the colon prefix.
- Working-Storage field.
- Number.
- Literal string enclosed in quotation marks or apostrophes.
- Subselect clause. To specify a Subselect, see below.

Specify Subselect clauses

To specify a subselect clause for a column, enter a value in the Operator field, and subselect in the Value(s) field to display the Subselect Specification screen, as shown in See Subselect Specification Screen. Alternatively, enter s(ubselect) in the selection field next to the column.

Figure 6-8. Subselect Specification Screen



On the Subselect Specification screen, perform the following steps:

a Select a record for the subselect by entering s next to the record in the From Record field.

b In the Column field, enter the column name for the subselect. Alternatively, enter s(elect) in the selection field next to the Column field to display a column list, select a column from the list, and press Enter to return to the Subselect Specification screen.

c In the Function field, you can enter a SQL function. Alternatively, enter s(elect) in the selection field next to the Function field to display a function list, select a function from it, and press PF3 to return to the Subselect Specification screen.

d In the Where Qualification field, you can qualify the subselect by entering values next to the Column, Oper, and Value fields.

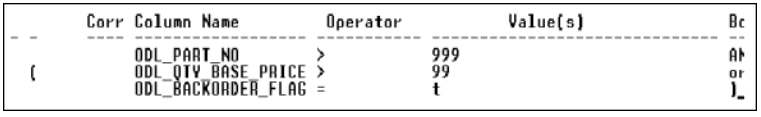
e Press Enter to preview the subselect clause as it will appear when generated. Then press PF3 to return to the SQL Qualification Specification screen.

Use Boolean qualifiers

To specify multiple conditions or value ranges for the call, enter the Boolean qualifier AND or OR in the Bool field. When you specify

two or more Boolean qualifiers in a call, you can group the qualification within parentheses as shown below:

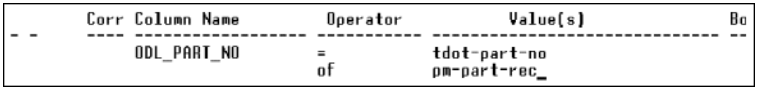
Figure 6-9. Grouping Qualifiers



Use the OF operator

If your qualification value is an elementary-level COBOL field that belongs to multiple group-level fields, insert a line and enter the OF operator and the group-level field to which it belongs. For example:

Figure 6-10. Using the OF Qualifier



Use Exists and Not Exists clauses

To specify an Exists clause, first insert a blank line below the call by entering i(nsert) in the selection field next to it. Leave the Column Name field blank, enter exists or not exists in the Operator field, and enter subselect in the Value field. Online Express displays the Subselect Specification screen, where you specify your Exists clause subselect criteria. Specify only one Exists clause per qualification. Press PF3 to return to the SQL Qualification Specification screen.

10 Press PF3 to exit the SQL Qualification Specification screen. APS displays one of the following screens, depending on the contents of your call:

If your call contains ...	**APS displays the ...**
No loop	Database Access Summary screen. The call definition is complete.
Loop with no Group By columns	Order By Columns screen. Perform step 12.
Loop with Group By columns	SQL Having Clause Specification screen. Perform step 11.

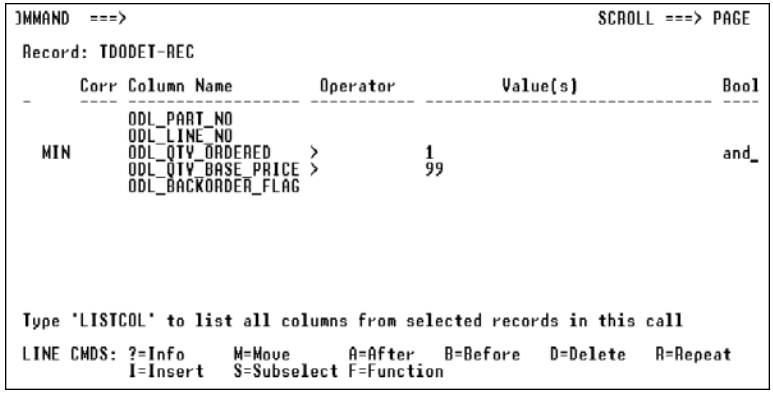
Define Having clauses

11 You can define a Having clause to qualify the Group By columns and columns to which you have assigned column functions in steps 7 or 9. To do so, select H(aving) on the Database Access Summary screen. The Having Clause Specification screen displays, as shown in *Grouping Qualifiers*, showing all such columns. Qualify them just as you qualify any column. In addition, you can assign column functions to a Having clause as follows:

- To assign the COUNT column function to test the number of rows found for the Group By columns, insert a line and enter * in the Column Name field, and appropriate values in the Operator and Value(s) fields. APS automatically displays the function name abbreviation CNT next to the clause.
- To assign any other column function, enter f(unction) in the selection field next to the clause to display a function list, and select a function from it.

After you define a Having clause, press PF3 to display the Order By Columns screen.

Figure 6-11. SQL Having Clause Specification Screen



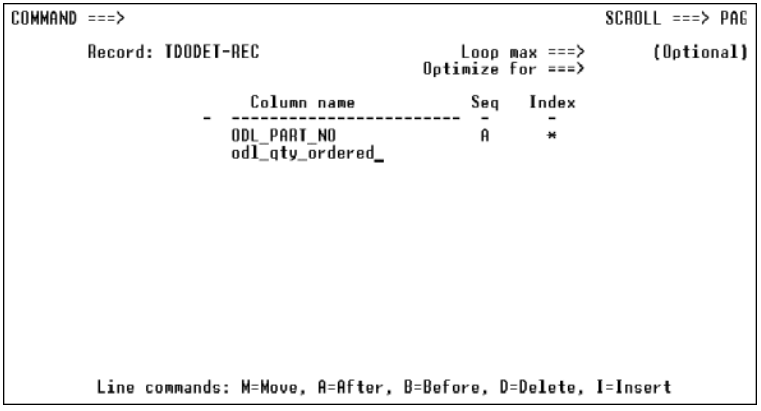
Order the columns

12 On the Order By Columns screen, specify the order in which the call obtains and displays the columns, as shown in *Using the OF Qualifier*. APS identifies the index column that you have selected, by displaying an asterisk in the Index field. If your subschema contains multiple indexes, APS displays only the last one listed in the subschema. To add any index or non-index columns to the list, insert a line and enter the column names. Alternatively, to display a

column selection list, enter listcol in the Command field and select columns from it.

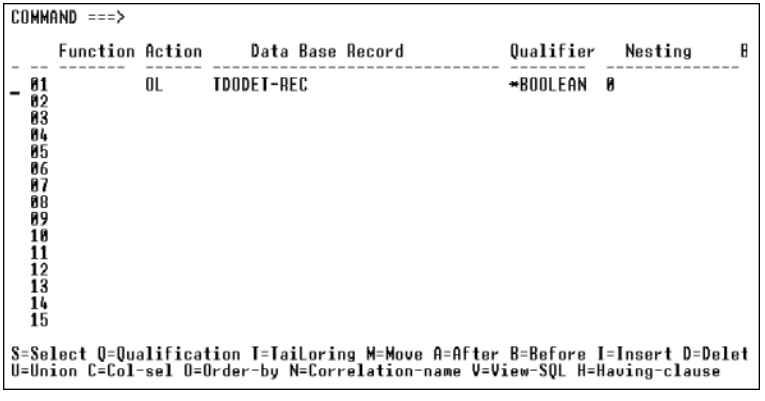
- If your call includes the modify or erase action codes, the index must be unique.
- To limit the number of rows that a loop call obtains, specify that number in the Optimize field.

Figure 6-12. Order By Columns Screen



13 Press PF3 to display the Database Access Summary screen. Your call definition is complete.

Figure 6-13. Database Access Summary Screen



Preview and test the call

14 You can preview the call definition as it will appear when generated, and test execute the call using SPUFI, the external interactive facility. To do so, follow these steps:

- On the Database Access Summary screen, preview the generated call definition by entering v(iew) in the selection field next to the call. The SQL Command Review screen displays, as shown in *Order By Columns Screen*. Note that the call is shown in the context of your program; several lines of your program source code precede the call as comments.

Figure 6-14. SQL Command Review Screen

```
COMMAND  ===> _                                              SCROLL ===> PAG
--******************************** TOP OF DATA ********************************
--
--     IDODET-01-READLOOP
--         IF OK-TO-PROCEED
--             EXEC SQL DECLARE PX01 CURSOR FOR
--             SELECT
                    ODL_PART_NO,
                    ODL_LINE_NO,
                    MIN(ODL_QTY_ORDERED),
                    ODL_QTY_BASE_PRICE,
                    ODL_BACKORDER_FLAG
                    FROM IDODET
                    WHERE
                    ODL_PART_NO = :IDOT-PART-NO AND
                    ODL_PART_NO IN
                    (SELECT
                    PM_PART_NO
                    FROM IDPART)
                    GROUP BY
                    ODL_PART_NO,
                    ODL_LINE_NO,
```

- Replace the call's host variables with any literal values, because SPUFI cannot use host variables.
- Enter save in the Command field to display the SQL Prototype screen.
- Select option 1, Save SQL, to save the generated call.
- To access SPUFI, select option 3, Invoke XDBSQL or SPUFI.

15 To define subsequent calls for a program, repeat the above steps. To modify any call definition, display the Database Access Summary screen and select the appropriate options displayed at the bottom of the screen. For example, to modify a call's qualification, enter the q (ualification) command in the selection field next to the call to display the SQL Qualification Specification screen.

16 When you finish defining all calls for your program, view a summary list of the calls by displaying the Database Access Summary screen. Ensure that the calls appear in the order in which you want them to execute. You can rearrange, add to, and delete from the list as follows:

- Move any call to a position before or after another call by typing m (move) next to the call, and either b (before) or a (after) next to another call.
- Add a call definition in between calls in the list by typing i (insert) next to a call and then defining the new call.
- Delete a call by typing d (delete) next to the call.

Defining Join Calls

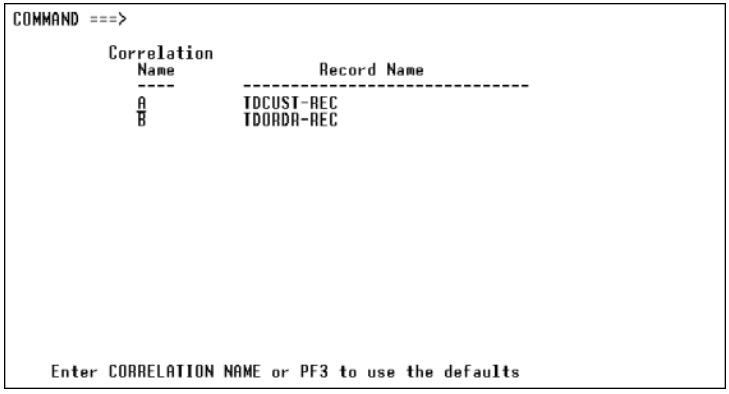
1 Select option 6, Database Access, from the Online Express menu. Alternatively, enter 6 or dba on any primary Online Express screen. The Database Access Summary screen displays.

2 Enter s in the selection field next to a new call number. The Database Record Selection screen displays, listing all tables in the program subschema.

3 Select up to 16 tables for the Join by entering the o(btain) action code and any other action codes next to the tables.

4 Press PF3 to display the Correlation Names screen, which shows default correlation names for each selected table. To override the default correlation names, simply overtype them. To reset the default names, enter reset in the Command field.

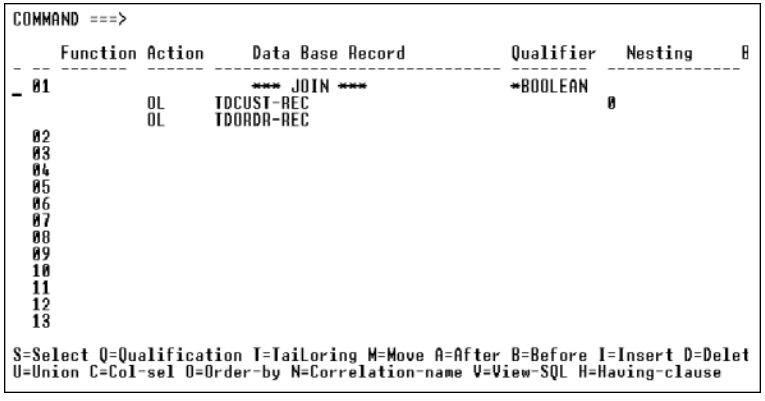
Figure 6-15. Correlation Names Screen



5 Press PF3 to display the Column Selection screen.

6 Perform steps 5 through 16 in *Defining Basic SQL Calls*. Your Join call definition is complete, as shown in *Database Access Summary Screen*.

Figure 6-16. Database Access Summary Screen



Defining Union Calls

Select the Database Access Summary screen

1 Select option 6, Database Access, from the Online Express menu. Alternatively, enter 6 or dba on any primary Online Express screen. The Database Access Summary screen displays.

Define the first Select statement

2 To define the first select statement of the Union, enter u(nion) next to a new call number on the Database Access Summary screen. APS displays the Database Record Selection screen.

3 Perform steps 3 through 9 in *Defining Basic SQL Calls*.

- Instead of selecting just one table on the Database Record Selection screen, select up to 16 tables for the Join by entering the o(btain) action code and any other action codes next to them.
- APS displays the Correlation Names screen, which contains a default correlation name for each selected table. To override the names, simply overtype them. To reset the default names, enter reset in the Command field. Press PF3 to display the Column Selection screen.

Define Having clauses

4 Press PF3 to exit the SQL Qualification Specification screen. APS displays one of the following screens, depending on whether your call contains Group By columns:

To define a Join within a Union, follow these steps:

If your call contains ...	**APS displays the ...**
Group By columns	SQL Having Clause Specification screen. Perform step 5.
No Group By columns	Union Summary screen. Perform step 6.

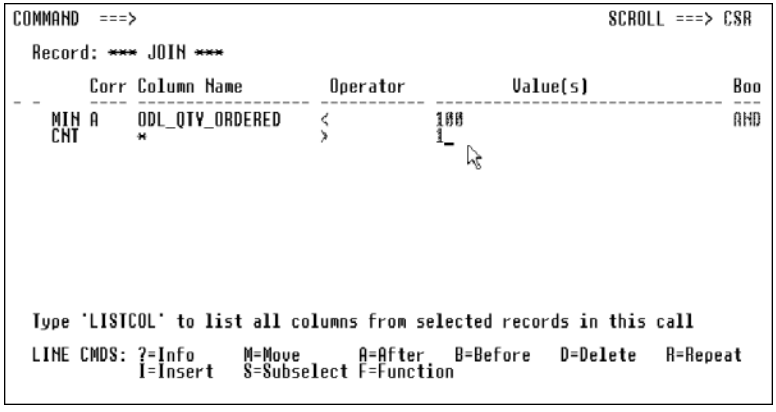
5 You can define a Having clause to qualify the Group By columns and columns to which you have assigned column functions in steps 7 or 9 in*Defining Basic SQL Calls*. The Having Clause Specification screen, shown in *Correlation Names Screen*, displays all such columns. Qualify them just as you qualify any column. In addition, you can assign column functions to a Having clause as follows:

- To assign the COUNT column function to test the number of rows found for the Group By columns, insert a line and enter * in the Column Name field, and appropriate values in the

Operator and Value(s) fields. APS automatically displays the function name abbreviation CNT next to the clause.

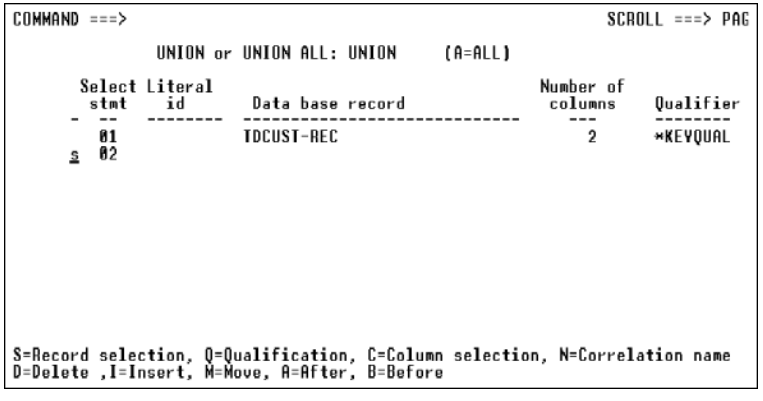
- To assign any other column function, enter f(unction) in the selection field next to the clause to display a function list, and select a function from it.

Figure 6-17. SQL Having Clause Specification Screen



6 After you define a Having clause, press PF3 to display the Union Summary Menu. The Menu displays the first Select statement that you just defined for the Union, as shown in *Union Summary Menu*.

Figure 6-18. Union Summary Menu



Define the next Select statement

7 On the Union Summary Menu, define the next Select statement by entering s in the selection field next to Select Stmt 02. APS displays the Database Record Selection screen. Repeat steps 3 through 6 above to define as many Select statements as you need for the Union. Online Express returns all column data to the host variables of the first Select statement.

Pad the Select statement columns

8 After you define all the Select statements, check the Number of Columns field to see whether each statement has an equal number of columns. If they do not, do the following:

- Ensure that the statement with the greatest number of columns is the first statement in the list. The other statements can be in any order. Remember that Online Express returns all column data to the host variables of the first Select statement.
- Ensure that the columns of each Select statement correspond to each other properly. If the last column in the first statement has no corresponding column, Online Express automatically pads the omitted column(s) with an appropriate value--either a blank character, a zero, DATE, TIME, or TIMESTAMP.

  For example, Online Express pads the third column of statement 2, below:

Stmt 1	Column 1	Column 2	Column 3
Stmt 2	Column 1	Column 2	
Stmt 3	Column 1	Column 2	Column 3

  If any column except the last column in the first statement has no corresponding column, you must pad the omitted column(s) on the Column Selection Update screen. For example, you must pad the second column of statement 2, below:

Stmt 1	Column 1	Column 2	Column 3
Stmt 2	Column 1		Column 3
Stmt 3	Column 1	Column 2	Column 3

- To pad any column, enter c(olumn selection) next to the Select statement to display the Column Selection Update screen. Insert a line between the appropriate columns and enter * in the Column Name field. Online Express pads the column with an appropriate value.

Ensure matching columns

9 After you define all the Select statements for the Union, press PF3 on the Union Summary Menu. If each column's corresponding column(s) match in data type and length, APS displays the Order By Columns screen; perform step 12. If any columns are mismatched, APS displays the mismatched columns on the Union Columns Cross Reference screen.

10 On the Union Columns Cross Reference screen, examine the data type and length of each column to find the error. Note which columns do not match, and press PF3 to return to the Union Summary Menu.

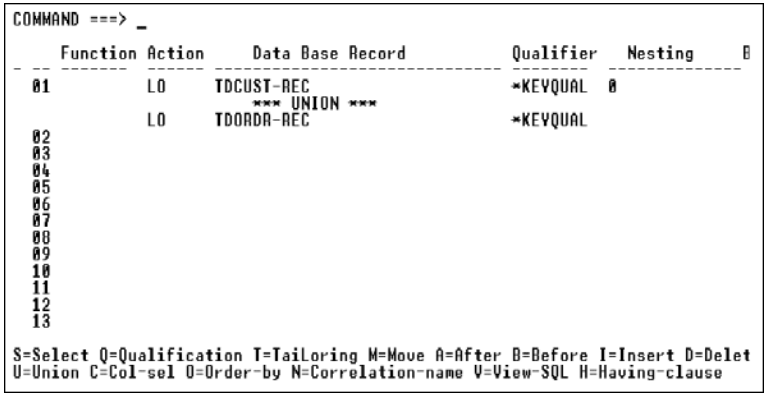
11 To correct the mismatch, enter c(olumn selection) in the selection field next to the Select statement that contains the mismatch. APS displays the Column Selection Update screen, where you can make the necessary changes. Then press PF3 to return to the Union Summary screen to ensure that the columns match now, and press PF3 to display the Order By Columns screen.

Order the columns

12 Specify the order in which the call obtains and displays the columns, as described in step 12 in *Defining Basic SQL Calls*.

13 Press PF3 to display the Database Access Summary screen. Your Union call definition is complete.

Figure 6-19. Database Access Summary Menu



Preview and test the call

14 To preview and test the call, see step 14 in *Defining Basic SQL Calls*.

Special Considerations

- While Online Express lets you completely define database calls without having to code them, you can also extend and customize those calls to suit your needs. See *Customized Database Calls*.
- If you specify multiple loops in your program, you must specify which loop that you plan to map to a repeated record block on your program screen. By default, Online Express assigns a 1 in the Blk (Block) field of the first loop on the Database Access Summary screen, and leaves the field blank for all other calls. If you plan to map fields of a different loop, enter the value 1 in its Blk field, and blank out the default Blk field value of the first loop. For more information, see *Nested Loops*.
- You can use COUNT(*) in Online Express. You code a COUNT(*) as follows:

  a Create a database call in Online Express.

  b From the database call summary type C in the line command to access the column selection.

  c Insert a line with 'I' in the line command field.

  d Under **Function** type COUNT; under **Column Name** type a asterisk (*); under **Cobol Name** type the name of the working storage field to receive the count, for example WS-COUNT.

  e When you generate the COBOL you get SQL similar to the following:

```
SELECT COUNT(*)  FROM TABLEA  INTO WS-COUNT.
```

Defining IMS Database Calls

Using IMS database calls

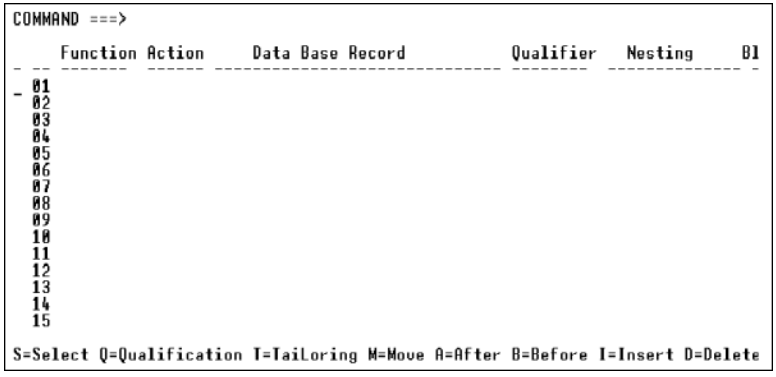
You can define IMS database calls to obtain, loop, modify, erase, and store any record. For example, you can obtain a parent record and loop on its child records to obtain multiple records that the end user can modify, erase, and store. Online Express stores a child record for the parent record that is currently obtained when the store action executes.

Follow these steps to define an IMS database call for any record:

Select the Database Access Summary screen

1 Select option 6, Database Access, from the Online Express menu. Alternatively, enter 6 or dba on any primary Online Express screen. The Database Access Summary screen displays.

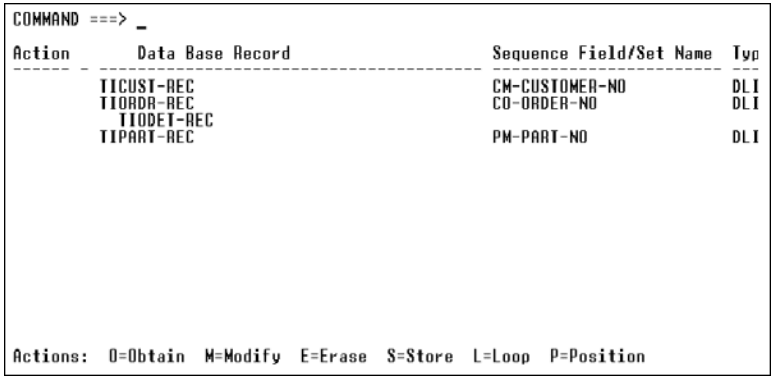
Figure 6-20. Database Access Summary Screen



Define the first call

2 To define the first call, enter s in the selection field next to call number 01. The Database Record Selection screen displays, listing all records in the program subschema. IMS parent and child records display, showing their parent/child relationships. Child records appear indented from their parents, as shown in *Database Record Selection Screen*.

Figure 6-21. Database Record Selection Screen



Specify actions

3 Enter one or more action codes, in any order, in the Action field next to the record of the first call. For example, *Specifying Database Action Codes* illustrates a simple call that obtains and loops on a parent record and allows the end user to modify, erase, and store it, as indicated by the o, l, m, e, and s action codes entered next to the record. To define nested loops, see *Nested Loops*.

Figure 6-22. Specifying Database Action Codes

```
COMMAND ===>

Action      Data Base Record                    Sequence Field/Set Name  Typ
------ - ---------------------------------------- ------------------------ ---
       TICUST-REC                                CM-CUSTOMER-NO           DLI
olmes_ TIORDR-REC                                CO-ORDER-NO              DLI
         TIODET-REC
       TIPART-REC                                PM-PART-NO               DLI
```

Obtain a child record

4 To obtain a child record of the parent that you just obtained, define another call to position the database pointer on the parent record and obtain the child. To do so, enter the p(osition) action code next to the parent record, and the o(btain) action code next to the child. To loop on the child, also enter the l(oop) action next to the parent. For example, to position on TIORDR-REC and loop on TIODET-REC, enter the p(osition) and l(oop) action codes next to TIORDR-REC, and the o(btain) action code next to TIODET-REC, as shown in *Positioning on a Parent Record to Obtain and Loop on Its Child*.

In addition, you can enter the m(odify), e(rase), and s(tore) action codes next to the child record if you want the end user to be able to perform those actions against it.

Figure 6-23. Positioning on a Parent Record to Obtain and Loop on Its Child

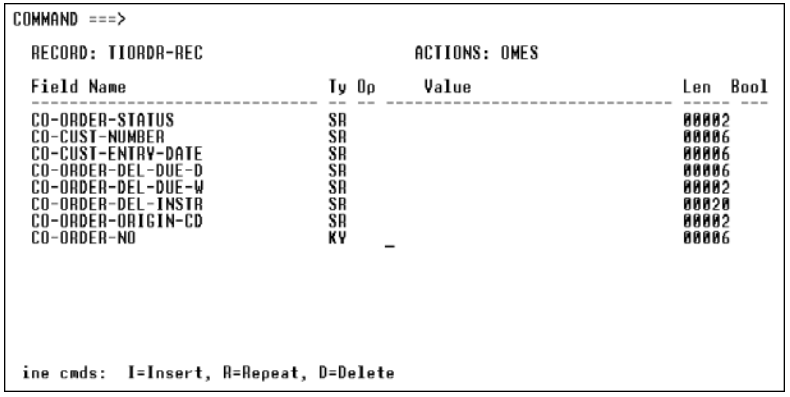
```
COMMAND ===>

Action      Data Base Record                    Sequence Field/Set Name  Typ
------ - ---------------------------------------- ------------------------ ---
       TICUST-REC                                CM-CUSTOMER-NO           DLI
pl     TIORDR-REC                                CO-ORDER-NO              DLI
omes_    TIODET-REC
       TIPART-REC                                PM-PART-NO               DLI
```

Qualify the record

5 From the current Database Record Selection screen, enter s in the selection field next to the record that you want to obtain. The

Database Qualification screen displays, listing all fields of the selected record, as shown in *Database Qualification Screen*.

Figure 6-24. Database Qualification Screen

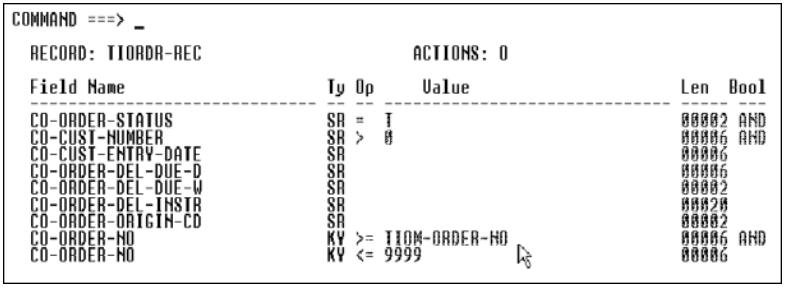


6 Qualify the record on one or more fields by entering an operator and a qualification value next to the field(s). A qualification value can be a COBOL screen field, Working-Storage field, a number, or a literal enclosed in quotes or apostrophes. To specify multiple conditions or value ranges for the call, enter the Boolean qualifier AND or OR. To let you specify Boolean qualification for a key field, APS copies the key field onto the next line. You can qualify the following types of fields described below. The field type for the field displays in the Ty(pe) field on the screen.

Field Type	Description
KY	Key field.
SQ	Sequence field of a child record's index set.
SR	Non-unique search field.

For example, in *Qualifying a Record* below, note that the call is qualified using Boolean qualification on three fields.

Figure 6-25. Qualifying a Record



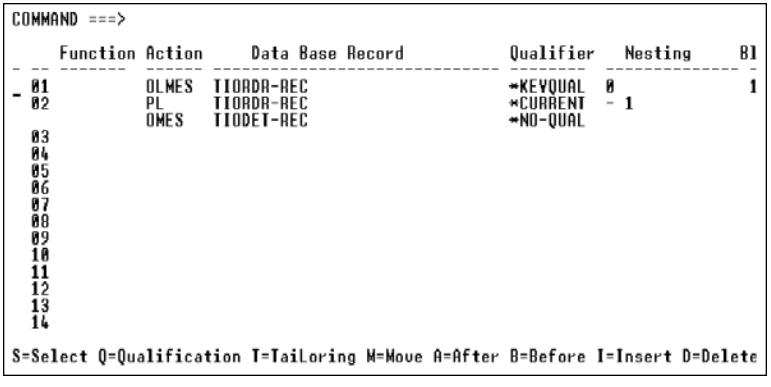
If your qualification value is a record field that must reference another field at a higher level in the hierarchy, insert a line and specify the OF operator in the Op field and the higher-level field in the Value field.

Save and review the specifications

7 When you finish qualifying the record, save your specifications.

8 View a summary of the call that you just defined by pressing PF3 twice. Note in *Database Access Summary Screen* that call 01 obtains, loops on, modifies, erases, and stores TIORDR-REC, qualified on its key field. Call 02 finds the currently-obtained TIORDR-REC and loops on its detail records, which the end user can modify, erase from, and store additional records with.

Figure 6-26. Database Access Summary Screen



Define additional calls

9 Repeat the above steps to define subsequent calls for a program, or to modify call definition.

View list of all calls

10 When you finish defining all calls for your program, view a summary list of the calls by displaying the Database Access Summary screen. Ensure that the calls appear in the order in which you want them to execute. Rearrange, add to, and delete from the list as follows:

- Move any call to a position before or after another call by typing m (move) next to the call, and either b (before) or a (after) next to another call.
- Add a call definition between calls in the list by typing i (insert) next to a call and then defining the new call.
- Delete a call by typing d (delete) next to the call.

Special Considerations

- While Online Express lets you completely define database calls without having to code them, you can also extend and customize those calls to suit your needs. See *Customized Database Calls*.
- If you specify multiple loops in your program, you must specify which loop you plan to map to a repeated record block on your program screen. By default, Online Express assigns the value 1 in the Blk (Block) field of the first loop on the Database Access Summary screen, and leaves the field blank for all other calls. If you plan to map fields of a different loop, enter the value 1 in its Blk field, and blank out the default Blk field value of the first loop. For more information, see *Nested Loops*.

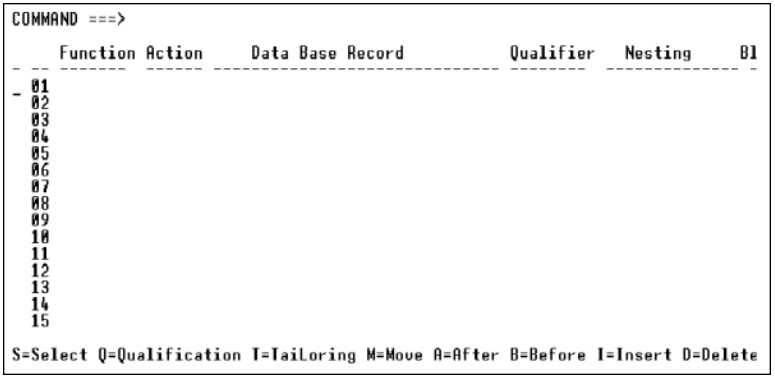
Defining VSAM Database Calls

You can define VSAM database calls to obtain, loop, modify, erase, and store any record. Follow these steps to define VSAM database calls:

Select Database Access

1 Select option 6, Database Access, from the Online Express menu. Alternatively, enter 6 or dba on any primary Online Express screen. The Database Access Summary screen displays.

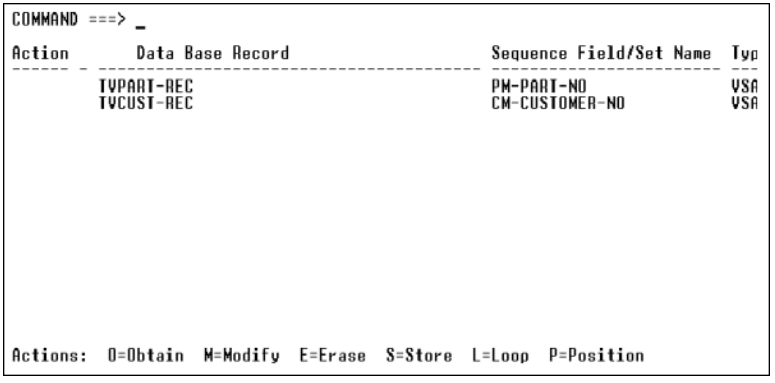
Figure 6-27. Database Access Summary Screen



Define the first call

2 To define the first call, enter s in the selection field next to call number 01. The Database Record Selection screen displays, listing all records in the program subschema, as shown in *Database Record Selection Screen*.

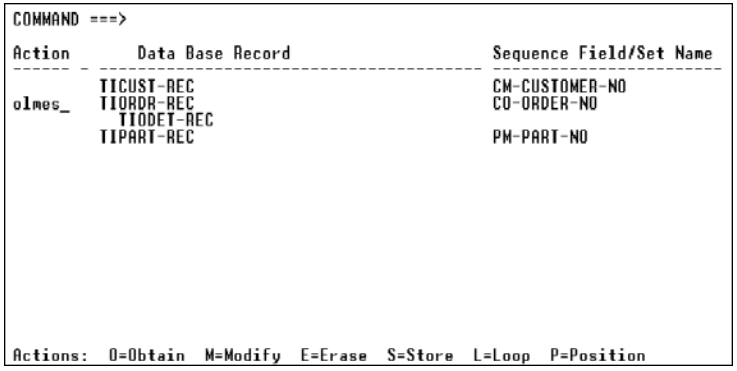
Figure 6-28. Database Record Selection Screen



Specify action codes

3 Enter one or more action codes, in any order, in the Action field next to the record of the first call. For example, *Specifying Database Action Codes* illustrates a simple call that obtains and loops on a record and allows the end user to modify, erase, and store it, as indicated by the o, l, m, e, and s action codes entered next to the record. To define nested loops, see *Nested Loops*.

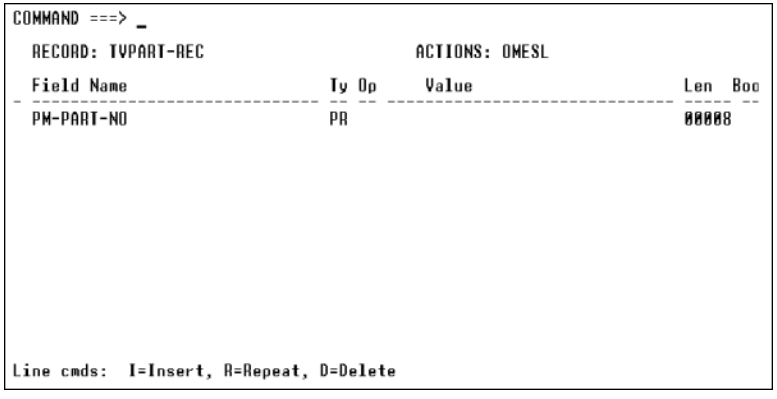
Figure 6-29. Specifying Database Action Codes



Qualify the record

4 Access the Database Qualification screen by entering s in the selection field next to the record that you want to obtain. The Database Qualification screen displays, listing all fields of the selected record, as shown in *Database Qualification Screen*.

Figure 6-30. Database Qualification Screen



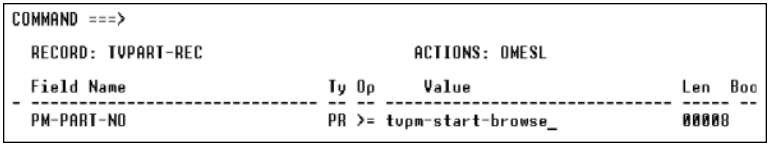
5 Qualify the record on one or more fields by entering an operator and a qualification value next to the field(s). A qualification value can be either a COBOL screen field or Working-Storage name, or a literal enclosed in quotation marks. You can qualify the following

types of fields, as shown below. Each field's type automatically displays in the Ty(pe) field on the screen.

Field Type	Description
KY	Key field. To qualify on a partial key, type over the value in the Len(gth) field.
PR	Primary index.
SR	Non-unique search field.

For example, in *Qualifying a Record*, the key field CO-ORDER-NO qualifies the record. The qualification is satisfied when the end user enters a value in the screen field PM-PART-NO is greater than or equal to the value in the database record TVPM-START-BROWSE.

Figure 6-31. Qualifying a Record



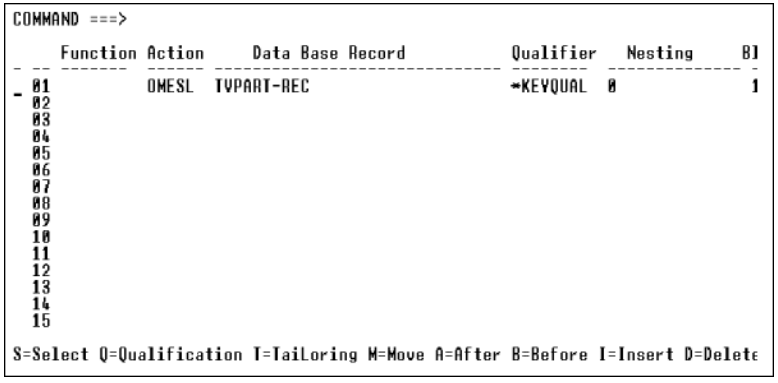
To qualify a record on a key field that consists of multiple fields, create a group-level qualification value field and move the values of the fields to it. You then qualify the key with the qualification value field. See *Multiple-Field Key Qualification*.

Save and review the specifications

6 When you finish qualifying the record, save your specifications.

7 View a summary of the call that you just defined by pressing PF3 twice. Note in *Database Access Summary Screen* that call 01 obtains, loops on, modifies, erases, and stores TVPM-START-BROWSE, qualified on its key field.

Figure 6-32. Database Access Summary Screen



Define additional calls

8 Repeat the above steps to define subsequent calls for a program, or to modify any call definition.

9 When you finish defining all calls for your program, view a summary list of the calls by displaying the Database Access Summary screen. Ensure that the calls appear in the order in which you want them to execute. You can rearrange, add to, and delete from the list as follows:

- Move any call to a position before or after another call by typing m (move) next to the call, and either b (before) or a (after) next to another call.
- Add a call definition in between calls in the list by typing i (insert) next to a call and then defining the new call.
- Delete a call by typing d (delete) next to the call.

Special Considerations

- While Online Express lets you completely define database calls without having to code them, you can also extend and customize those calls to suit your needs. See *Customized Database Calls*.
- If you specify multiple loops in your program, you must specify which loop that you plan to map to a repeated record block on your program screen. By default, Online Express assigns the value 1 in the

Blk (Block) field of the first loop on the Database Access Summary screen, and leaves the field blank for all other calls. If you plan to map fields of a different loop, enter the value 1 in its Blk field, and blank out the default Blk field value of the first loop. For more information, see *Nested Loops*.

Defining IDMS Database Calls

Using IDMS database calls

You can define IDMS database calls to obtain, loop, modify, erase, and store any record-an owner record, a member record, or an independent record that is neither an owner or member. For example, you can obtain an owner record and loop on its member records to obtain multiple records that the end user can modify, erase, and store. Online Express stores a member record for the owner record that is currently obtained when the store action executes.

Connecting and disconnecting records in owner/member sets

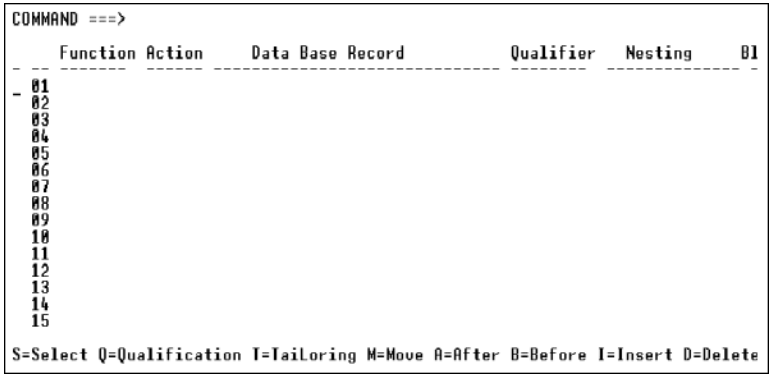
In addition, you can connect and disconnect records in owner/member sets. For example, you might want to change ownership of an employee record from one department record to another. To do so, you use action codes to disconnect the employee record from its current department record, and connect it to a different department record. Or you might want to disconnect the ownership of the employee record completely, making it an independent record. All records in a disconnect/connect operation must be current of record type, meaning that they must be obtained immediately before the operation executes.

Follow these steps to define an IDMS database call for any record:

Select Database Access

1 Select option 6, Database Access, from the Online Express menu. Alternatively, enter 6 or dba on any primary Online Express screen. The Database Access Summary screen displays.

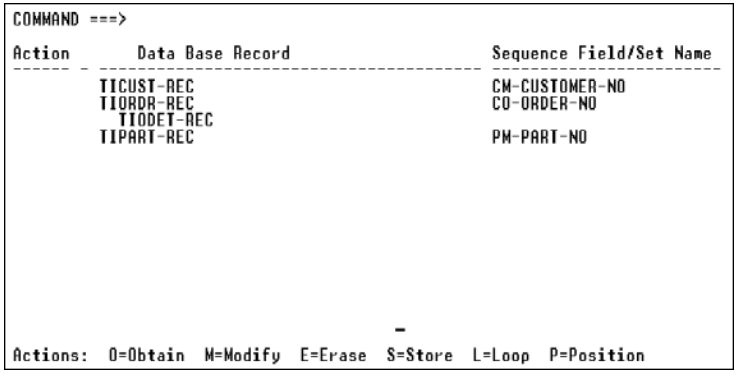
Figure 6-33. Database Access Summary Screen



Define the first call

2 To define the first call, enter s in the selection field next to call number 01. The Database Record Selection screen displays, listing all records in the program subschema. IDMS owner and member records display, showing their owner/member set relationships. Member records appear indented from their owners, as shown in *Database Record Selection Screen*.

Figure 6-34. Database Record Selection Screen

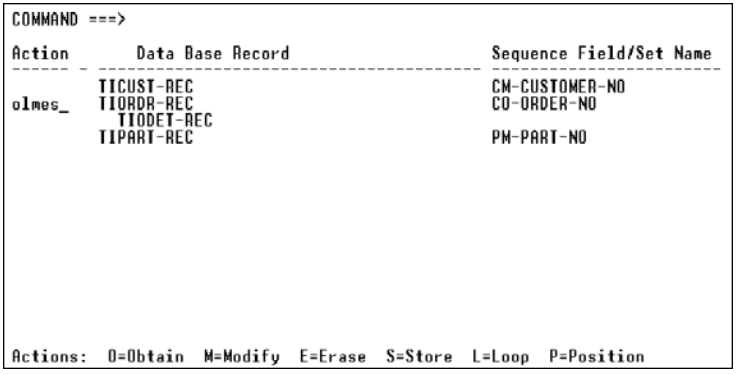


Specify actions

3 Enter one or more action codes, in any order, in the Action field next to the record of the first call. For example, *Specifying Database Action Codes* illustrates a simple call that obtains and loops on an owner record and allows the end user to modify, erase, and store it,

as indicated by the o, l, m, e, and s action codes entered next to the record. To define nested loops, see *Nested Loops*.

Figure 6-35. Specifying Database Action Codes

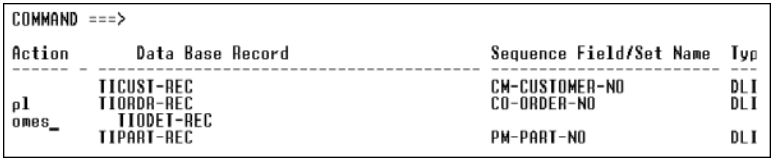


Obtain member records

4 To obtain a member record of the owner that you just obtained, define another call to find the owner record and obtain the member. To do so, enter the p action code next to the owner record, and the o(btain) action code next to the member. To loop on the member, also enter the l(oop) action next to the owner. For example, to find TIORDR-REC and loop on TIODET-REC, enter the p and l(oop) action codes next to TIORDR-REC, and the o(btain) action code next to TIODET-REC, as shown in *Finding an Owner Record to Obtain and Loop on Its Member*.

In addition, you can enter the m(odify), e(rase), and s(tore) action codes next to the member record if you want the end user to be able to perform those actions against it.

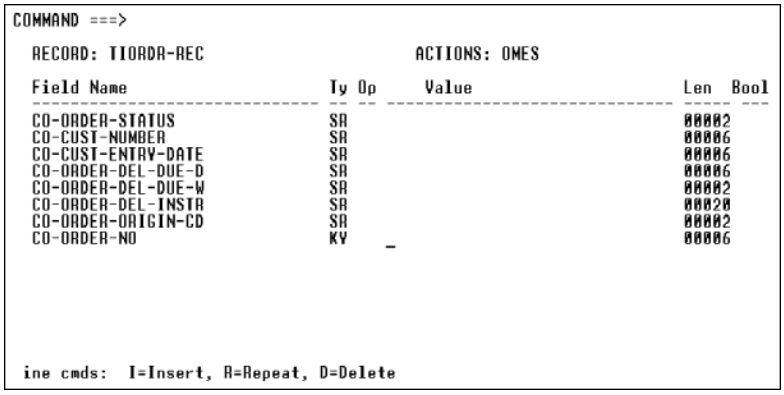
Figure 6-36. Finding an Owner Record to Obtain and Loop on Its Member



Qualify the record

5 From the current Database Record Selection screen, access the Database Qualification screen by entering s in the selection field next to the record that you want to obtain. The Database Qualification screen displays, listing all fields of the selected record, as shown in *Database Qualification Screen*.

Figure 6-37. Database Qualification Screen



6 Qualify the record on one or more fields by entering an operator and a qualification value next to the field(s). A qualification value can be either a COBOL screen field or Working-Storage name, or a literal enclosed in quotation marks. You can qualify the following types of fields, as shown below. Each field's type automatically displays in the Ty(pe) field on the screen.

Field Type	Description
AD	Address field, if one exists. Online Express displays the address field as a field named DB-KEY.
CA	CALC key field.
KY	Key field.
SQ	Sequence field of a member record's index set.
SR	Non-unique search field.

For example, in *Qualifying a Record* below, the key field CO-ORDER-NO qualifies the record. The qualification is satisfied when the end user enters a value in the screen field TIOM-ORDER-NO that equals a value in the database record field CO-ORDER-NO.

Figure 6-38. Qualifying a Record

```
COMMAND ===>

  RECORD: TIORDR-REC                        ACTIONS: OMES

  Field Name                      Ty Op     Value                      Len  Ba
- ------------------------------ -- -- ------------------------------ ----- -
  CO-ORDER-STATUS                 SR                                   00002
  CO-CUST-NUMBER                  SR                                   00006
  CO-CUST-ENTRY-DATE              SR                                   00006
  CO-ORDER-DEL-DUE-D              SR                                   00006
  CO-ORDER-DEL-DUE-W              SR                                   00002
  CO-ORDER-DEL-INSTR              SR                                   00020
  CO-ORDER-ORIGIN-CD              SR                                   00002
  CO-ORDER-NO                     KV =  tiom-order-no_                 00006
```

To qualify a record on a key field that consists of multiple fields, create a group-level qualification value field and move the values of the fields to it. You then qualify the key with the qualification value field. See *Multiple-Field Key Qualification*, later in this chapter.

Save and review the specifications

7 When you finish qualifying the record, save your specifications.

8 View a summary of the call that you just defined by pressing PF3 twice. Note in *Database Access Summary Screen* that call 01 obtains, loops on, modifies, erases, and stores TIORDR-REC, qualified on its key field. Call 02 finds the currently-obtained TIORDR-REC and loops on its detail records, which the end user can modify, erase from, and store additional records with.

Figure 6-39. Database Access Summary Screen

```
COMMAND ===>

Action       Data Base Record                    Sequence Field/Set Name
------ - ------------------------------------ ------------------------
         TICUST-REC                             CM-CUSTOMER-NO
         TIORDR-REC                             CO-ORDER-NO
           TIODET-REC
         TIPART-REC                             PM-PART-NO

                                        -
Actions:  O=Obtain  M=Modify  E=Erase  S=Store  L=Loop  P=Position
```

9 Repeat the above steps to define subsequent calls for a program, or to modify any call definition.

10 When you finish defining all calls for your program, view a summary list of the calls by displaying the Database Access Summary screen. Ensure that the calls appear in the order in which you want them to execute. You can rearrange, add to, and delete from the list as follows:

- Move any call to a position before or after another call by typing m (move) next to the call, and either b (before) or a (after) next to another call.
- Add a call definition in between calls in the list by typing i (insert) next to a call and then defining the new call.
- Delete a call by typing d (delete) next to the call.

Special Considerations

- While Online Express lets you completely define database calls without having to code them, you can also extend and customize those calls to suit your needs. See *Customized Database Calls*.
- If you specify multiple loops in your program, you must specify which loop that you plan to map to a repeated record block on your program screen. By default, Online Express assigns the value 1 in the Blk (Block) field of the first loop on the Database Access Summary screen, and leaves the field blank for all other calls. If you plan to map fields of a different loop, enter the value 1 in its Blk field, and blank out the default Blk field value of the first loop. For more information, see *Nested Loops*.

Connecting and Disconnecting Records

You can define database calls that connect and disconnect records to and from owner/member sets. To do so, perform the following steps:

1 Ensure that you have defined the Update function in your program definition and on the program screen.

2 Define a call that obtains the member record that you want to disconnect by entering the o(btain) action code next to it.

3 Define a call that finds the member's owner record by entering the p action code next to the owner.

4 Define a call that disconnects the member from its current owner record by entering the d(isconnect) action code next to the member.

5 Define a call that obtains the new owner record that you want to connect the member record to.

6 Define a call that connects the member record to the new owner by entering the c(onnect) action code next to the member.

7 Display the Database Access Summary screen to view all calls that you created in the above steps, as shown in *Database Access Summary of Connect/Disconnect Program*.

Database Access Summary of Connect/Disconnect Program:

```
   Function Action Data Base Record      Qualifier Nesting
-- -------- ------ --------------------- --------- -------
01 *UPDATE  O      EMPLOYEE-REC           *KEYQUAL  0
02 *UPDATE  P      DEPT-A-REC             *NO-QUAL  0
03 *UPDATE  D      EMPLOYEE-REC           *NO-QUAL  0
04 *UPDATE  O      DEPT-B-REC             *KEYQUAL  0
05 *UPDATE  C      EMPLOYEE-REC           *NO-QUAL  0
```

8 Enter *update in the Function field of each call to update the modified owner/member relationship, as shown in *Database Access Summary of Connect/Disconnect Program*. The *update entries cause the program to execute all the calls when the end user executes the Update function on the program screen.

Customized Database Calls

Six basic tailoring options

While Online Express lets you completely define database calls without having to code them, you can also extend and customize those calls to suit your needs. Use any of the following techniques, in any of the supported database environments:

- Define nested loop calls.
- Execute multiple database actions with one program function.

- Write and execute custom processing routines at APS-provided locations in your program known as database call control points. Control points let you add customized error processing routines and routines that you want to execute before or after a database call.
- Override status codes and error messages.
- Qualify a record key that consists of multiple fields.
- Execute a call as a custom program function anywhere that you can perform a paragraph, such as at a program control point.

The following sections explain these techniques.

Nested Loops

Using nested loops

You use nested loops to obtain multiple occurrences of multiple records. Loops that are not nested obtain multiple occurrences of a single record. You can map any loop records to a repeated record block, list box, or combination box on your screen, or you can loop on records to calculate field totals, and display just the calculation results.

Nesting levels

When you define two loop calls in a program, the first loop is an outer loop, and the second loop is an inner, or nested, loop. The nested loop executes repeatedly each time that the outer loop executes once, and obtains all records that satisfy the outer loop record key. For example, you might want to loop on all order records of a particular customer, and loop on all detail records of all the order records.

Or you might want to loop on a record to obtain a certain record occurrence that you loop on again, and display just the second loop's records. For example, if your program must display all items to be included in the next shipment to a certain customer, you first loop on all outstanding orders for the customer to determine which order ships next. You then loop on that order to obtain all its detail records. *Executing the Store Action When Querying a Record* and *Database Call Tailoring Screen* illustrate this example.

To indicate that the inner loop is nested within the outer loop, Online Express assigns the default nesting level value - 1 to it on the Database Access Summary screen. The outer loop's nesting level is 0, indicating that it is not nested.

When you define more than two loops, each loop is nested within the previous loop. That is, the second loop is nested within the first, and the third loop is nested within the second. The default nesting level of the third loop is - - 2. You can define as many nested loops in your program as you need.

Example of nested loop calls

In the sample IMS program in *A Nested Loop*, a nested loop obtains all detail records of each order record that is obtained by the program's previous loop. In addition, the call following the nested loop obtains the part master record for each detail record. Fields from all three records display in a repeated record block on the program screen.

Figure 6-40. A Nested Loop

```
COMMAND ===> _

     Function Action     Data Base Record           Qualifier   Nesting     B
- -- -------- ------ ------------------------------ ---------  ------------
  01          O      TICUST-REC                     *KEYQUAL   0
  02          OL     TIORDR-REC                     *KEYQUAL   0
  03          PL     TIORDR-REC                     *CURRENT   - 1
              O      TIODET-REC                     *NO-QUAL
  04          O      TIPART-REC                     *KEYQUAL   - - 2
  05
```

The sample program above executes as follows:

- Call 01 obtains the customer record. It is qualified on the customer record key, the customer number.
- Call 02 is the outer loop. It loops on the order record, qualified on the order record key, the customer number. When the outer loop finds the first order, call 03 executes.
- Call 03 is the nested loop. It positions the database pointer on the currently-obtained first order and loops on the detail record. When it finds the detail record that is associated with the first order, call 04 executes.
- Call 04 is an obtain call, nested within the second loop. It obtains the part master record associated with the detail record that was obtained by the nested loop.
- Calls 03 and 04 execute repeatedly until no more detail and part master records are found for the first order record.
- Call 02 executes again, obtaining the second order record.

- Calls 03 and 04 execute repeatedly until no more detail and part master records are found for the second order record.
- Calls 02 through 04 execute repeatedly until no more order, detail, or part master records are found for the customer number specified in the customer record call, call 01.

Overriding nesting levels

Depending on what you want your program to do, you might need to override the nesting levels of nested loops to nest them under different loops, or to execute them independently of other loops. By default, Online Express assigns default nesting levels to each loop, in the Nesting field on the Database Access Summary screen. If you define three loops in your program, their default Nesting field values are as follows:

Loop	Nesting Value	Description
1	0	Loop is not nested
2	- 1	Loop is nested within loop 1
3	- - 2	Loop nested within loop 2

You can decrease the default nesting level of any loop simply by typing over the Nesting field value. For example, if you want the second loop to execute independently of the first loop, you change its nesting level from - 1 to 0. Or, if you want the third loop to be nested within the first loop rather than the second, you change its nesting level from - - 2 to - 1. APS ensures that your nesting levels do not skip a level. For example, you cannot specify a level 0 loop, followed by a level - - 2 loop.

When you override nesting levels, ensure that the value 1 appears in the Blk (Block) field of the outer loop that you plan to map to a repeated record block on your program screen. For example, if you have two loops with the nesting level 0, and you want to map fields of only the second loop to your screen, blank out the default value 1 in the Blk field of the first loop, and enter 1 in the Blk field of the second loop.

Example of overridding nesting levels

In the following example, the program must display all items to be included in the next shipment to a certain customer. The first loop reads all outstanding orders for the customer and executes a user-defined routine to determine which order ships next. The second loop obtains all detail records of the order found by the first loop, and maps them to a repeated record block.

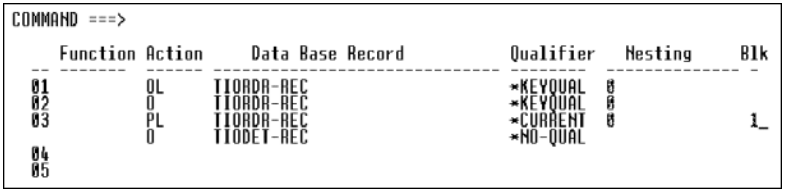
Note that if the default nesting level of the second loop is used, the program would not execute as required. The program would loop on and obtain all customer order records and their detail records. In

addition, Online Express would assume that records of the first loop will map to the repeated block.

To make the program execute as required, note in *Overridden Nesting Level and Blk Values* that:

- The nesting level of calls 02 and 03 are changed from - 1 to 0 to make them not nested within the first loop.
- The Blk field value of the first loop is changed from 1 to spaces to indicate that the loop does not display in the repeated record block.
- The Blk field value of the second loop is changed from spaces to 1 to indicate that the loop does display in the repeated record block.

Figure 6-41. Overridden Nesting Level and Blk Values

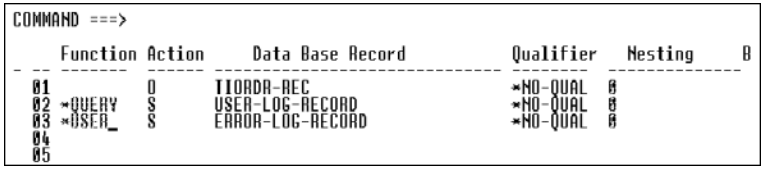


Functions with Multiple Database Actions

You can define a program function to execute more than one database action. For example, you can define the query function to execute the store action as well as the obtain action.

Suppose you want to store in a log record the ID of each end user who queries a customer order record. You define one call to obtain the order record, and another call to store the IDs in another record, as shown in *Executing the Store Action When Querying a Record*. To cause the query function to execute the store call as well as the obtain call, you enter the value *query in the Function field next to the store call. You then customize the store call with user-defined logic to move the user IDs to the log record. To write and execute custom logic for database calls, see *Custom Logic at Database Call Control Points*.

Figure 6-42. Executing the Store Action When Querying a Record



Custom Logic at Database Call Control Points

Write custom logic for database calls

Without leaving Online Express, you can write and automatically execute custom database call processing logic to supplement or override the default logic that Online Express generates. You execute custom logic at any of several APS-provided locations in your program, known as database call control points. The control points let you add processing logic before and after a database call, and when calls execute normally or abnormally. You select control points from a list that displays on the Database Call Tailoring screen. The list includes the following control points:

Control Point	Location
Befor DB Access	Before a non-loop database call executes
Before Loop	Before a loop database call executes
Normal Status (Before Record is Processed)	Before Online Express maps looped records to the screen
Normal Status	After Online Express maps any records to the screen
Exception Status	After the database call returns a status flag with the Exception status code
Error Status	After the database call returns a status flag with the Error status code
After DB Access	After a non-loop database call executes
After Loop	After a loop database call executes

Write local or global custom processing logic

You can write control point logic specifically for one program, or for use throughout your application. Program-specific custom logic is known as a local program stub; custom logic that you use throughout your application is known as a global program stub. Alternatively, you can write a macro and invoke it in any program of any application. You execute any stub or macro at any control point.

Local stubs

A local stub can consist of Procedure Division and Data Division code. You write a local stub in the Specification Painter, which you access from the Database Call Tailoring screen.

Global stubs

A global stub can consist of Procedure Division paragraphs. You write a global stub in the Program Painter, which you access from the Application Painter.

User-defined macros

A macro can consist of any code that you write using the APS Customization Facility, a high-level tool for writing and processing macros. You include macro library members in your application on the Application Painter screen.

Tailor individual database call actions

You add custom logic to, or tailor, each action of a database call individually. For example, you might want to tailor the obtain action by adding a data validation routine that executes whenever the obtain action executes.

The Normal Status control points

The Normal Status (Before Record is Processed) control point lets you add custom logic before looped records map to a repeated record block on your screen. Use this control point if you want to map only some of the records that a loop obtains. In your stub or macro, write conditional logic to determine which records to map. Online Express provides a flag, OK-TO-PROCEED, that you set to True to map and process the record, or False to bypass mapping and processing. You can ignore the flag if you do not use this control point; the flag is set to True by default. To add custom logic after Online Express maps any record to your screen, use the Normal Status control point.

The following example illustrates both control points. Suppose that you must map the records that show annual sales of $100,000 or more in the Northwest region, and calculate and map the grand total of those records. You first define a loop call and qualify it to obtain the records of $100,000 or more. Then you tailor the loop call with two local stubs.

The first stub checks the records obtained by the loop to allow only records of the Northwest region to be processed further. The second stub calculates the grand total of those records, and maps the total to

the screen. The generated loop call and stub paragraphs are shown below:

```
DB-PROCESS REC SALES-RECORD
... WHERE ANNUAL-SALES-TOTAL > 99999
    PERFORM CHECK-BEFORE-MAPPING-STUB-PARA
    IF OK-TO-PROCEED
        ADD 1 TO CTR
        PERFORM RECORD-STOREKEY-PARA
        MOVE REC-TO-SCREEN-BLK1
        PERFORM CHECK-AFTER-MAPPING-STUB-PARA
.
.
.
CHECK-BEFORE-MAPPING-PARA
    TRUE OK-TO-PROCEED
    IF SALES-REGION NOT = NORTHWEST
        FALSE OK-TO-PROCEED

CHECK-AFTER-MAPPING-PARA
    calculation and mapping routine for grand total
```

Note that:

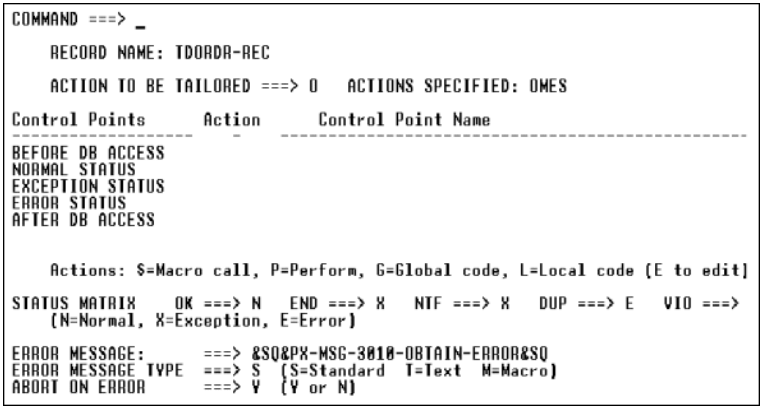
- Online Express generates the loop call as an APS DB-PROCESS call.
- The CHECK-BEFORE-MAPPING paragraph is written and executes at the Normal Status (Before Record is Processed) control point. Online Express generates the paragraph's PERFORM statement.
- The CHECK-AFTER-MAPPING paragraph is written and executes at the Normal Status control point. Online Express generates the paragraph's PERFORM statement.
- Online Express generates all other lines of code that are subordinate to the DB-PROCESS call.

Adding Custom Logic To a Call

Write and automatically execute custom logic for a call as follows :

1. Display the call on the Database Access Summary screen.
2. Enter t(ailoring) next to the call to display the Database Call Tailoring screen.
3. Specify which action that you want to customize by entering its action code, such as o(btain) or s(tore), in the Action To Be Tailored field, as shown in *Database Call Tailoring Screen*. You can tailor the Obtain, Modify, Store, and Erase actions.

Figure 6-43. Database Call Tailoring Screen

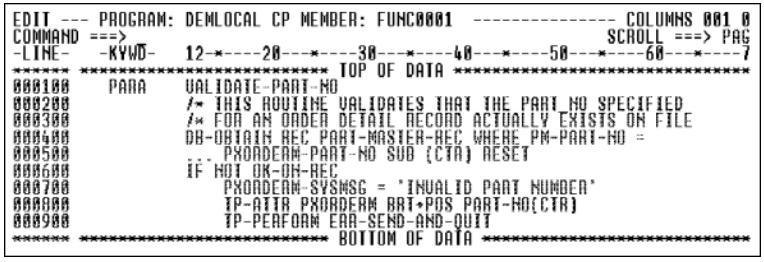


4 In the Action field next to the control point where you want to add logic, either invoke a macro that contains the logic, execute a global stub that contains the logic, or write and execute the logic in a local stub, as follows:

- To invoke a macro, enter $ in the Action field, and the macro name in the Control Point Name field. The macro must reside in the USERMACS library member that you specify on the Application Painter screen. For rules on writing macros, see the APS Customization Facility User's Guide.
- To execute a global stub, enter g in the Action field, and the global stub name in the Control Point Name field. You must define the global stub in the Program Painter and specify its name on the Application Painter screen. For rules on writing global stubs, see *Custom Program Functions*.
- To write and execute a local stub, perform steps 5 and 6.

5 To write a local stub, first enter e(dit) in the Action field next to the control point where you want to write the logic. The Specification Painter displays, as shown in *Error Flag Status Codes*.

Figure 6-44. Writing a Local Stub in the Specification Painter



6 Write the local stub in the Specification Painter and save it. For rules on writing local stubs, see *Defining Custom Program Functions*. You do not name a local stub. After you save the stub, Online Express redisplays the Database Call Tailoring screen with the message PAINTED next to the control point.

Status Codes and Error Messages

You can customize database call processing to override the status codes of Online Express status flags and the text of default error messages. You do both on the Database Call Tailoring screen.

Online Express provides five status flags. By default, all status flags except OK-ON-REC return the Error status code, as shown below:

Status Flag	Default Status Code
OK-ON-REC	N(ormal)
END-ON-REC	E(rror)
NTF-ON-REC	E(rror)
DUP-ON-REC	E(rror)
VIO-ON-REC	E(rror)

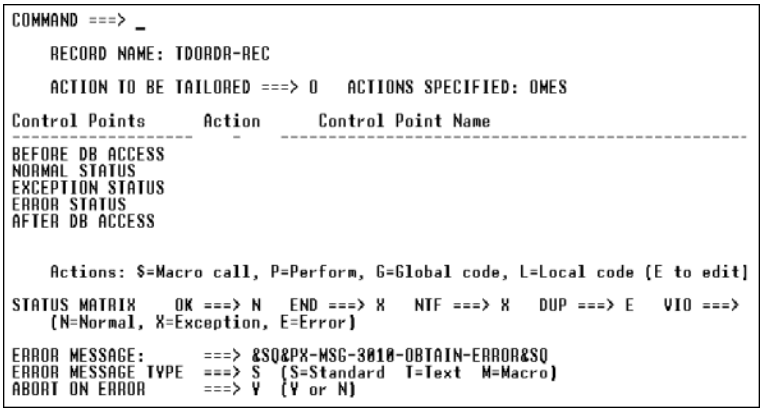
When Online Express returns the Error status flag, the program aborts and performs the Error-Send-And-Quit paragraph.

Overriding status codes

To override the default Error flag processing, you can change a status flag's status code from Error to Exception, and then write your own error routines at control points on the Call Tailoring screen. You do so

by overtyping the status code values in the Status Matrix fields, as shown in *Error Flag Status Codes* and writing error routines as described in *Custom Logic at Database Call Control Points*. To just prevent the Error flag from aborting the program, specify n for the Abort On Error field on the Database Call Tailoring screen.

Figure 6-45. Error Flag Status Codes



Overriding error messages

Online Express generates error messages that show which type of call failed and which record caused the failure. You can override the default messages with either a text message or a macro that contains a text message. To do so, enter either the text or the macro name in the Error Message field, and specify in the Error Message Type field whether you entered text or a macro.

Multiple-Field Key Qualification

Qualify group-level keys

To qualify a VSAM or IDMS group-level key field, you write custom logic that moves the key's elementary field values to a group-level qualification value field that you define. You then qualify the key with the qualification value field. For example, suppose that a key has the following elementary fields:

```
01  SALES-KEY.
    05  REGION-CODE    PIC X(2).
    05  YEAR-CODE      PIC X(2).
```

You write custom logic that defines a group-level Working-Storage field and moves the values of the two elementary fields to it, as follows:

```
-KYWD-  12-*----20---*----30---*----40---*----50---*--
        MOVE SCREEN-REGION-CODE TO REGION-CODE
        MOVE SCREEN-YEAR-CODE TO YEAR-CODE
 WS01   NEW-SALES-KEY.
        05  REGION-CODE    PIC X(2).
        05  YEAR-CODE      PIC X(2).
```

On the Database Qualification screen, you qualify the key field, SALES-KEY, with the qualification value field, NEW-SALES-KEY.

Write and execute the custom logic

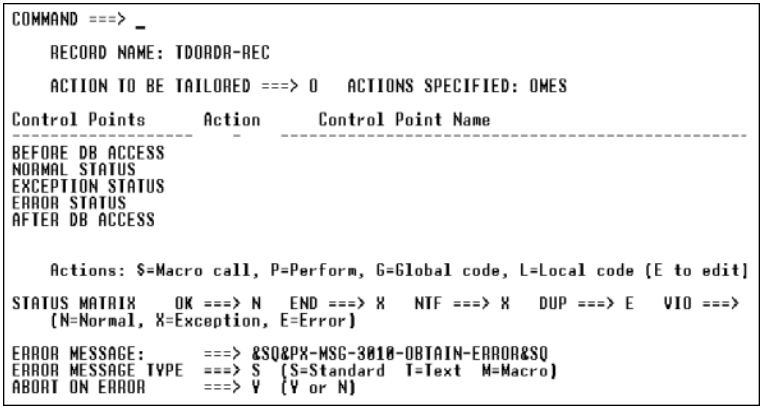
You write and execute the custom logic at the Before DB Access control point, on the Database Call Tailoring screen. See *Custom Logic at Database Call Control Points*.

Qualifying Multiple-Field Keys

Follow these steps to qualify a multiple-field key field with a qualification value field:

1. Define the call as you normally do, using the Database Record Selection screen, but do not qualify the call yet.
2. Return to the Database Access Summary screen.
3. On the Database Access Summary screen, enter t(ailoring) next to the call to display the Database Call Tailoring screen.
4. If the o(btain) action does not display in the Action To Be Tailored field, enter it now, as shown in *Database Call Tailoring Screen for the Obtain Action*.

Figure 6-46. Database Call Tailoring Screen for the Obtain Action



5 Enter e(dit) in the Action field next to the control point named Before DB Access. The Specification Editor displays so that you can write the qualification value field logic in a local stub. Online Express automatically executes the local stub at this control point, before the call executes. See *Defining Custom Program Functions*, for rules on coding local stubs. Alternatively, you can execute a global stub or invoke a user-defined macro at the control point.

6 Access the Database Qualification screen.

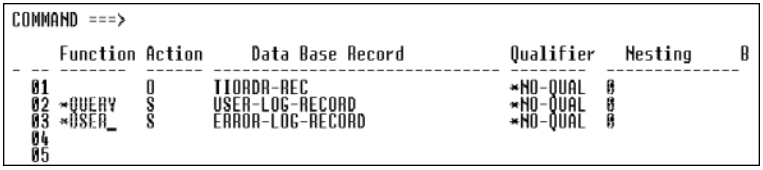
7 Qualify the key field by entering the = operator in the Operator field, and the group-level qualification value field in the Value field. In this example, you qualify the key field SALES-KEY with the value field NEW-SALES-KEY.

Database Calls as Custom Program Functions

Typically, you define calls that execute when the end user enters a function code, presses a key. You can define additional calls that you execute anywhere that you can execute a paragraph, such as at a control point. For example, you might want to store in a log record the error messages that end users receive when querying a customer order record. You first define a call that stores the log record. Then, on the

Database Access Summary screen, you enter *user in the Function field to indicate that you will execute the call as a paragraph, somewhere in Online Express, as illustrated in Call 03 in *Defining a Call That You Execute As a Custom Function*:

Figure 6-47. Defining a Call That You Execute As a Custom Function



After you define the *user call, Online Express writes it to a paragraph. You then execute the paragraph anywhere that Online Express allows, such as at a control point or on the Alternate Functions screen. In the example above, you would execute the paragraph at the control point after the screen is read, named POST-SCREEN-READ, on the Control Points screen. You then would write and execute custom code at the AFTER DB ACCESS control point on the Database Call Tailoring screen to move the error messages to the log record.

Follow these steps to write and execute a call in a custom program function:

1. Define the call as you normally do. On the Database Access Summary screen, enter *user in the Function field next to the call.
2. View the name of the paragraph to which Online Express writes the *user call. To do so, enter *user in the Command field. The User Controlled Database Calls screen displays, showing the APS-generated paragraph name. You can override the name on this screen by overtyping it.
3. Perform the paragraph anywhere that Online Express allows.

Savekey and Commarea Storage

Purpose of savekey storage

You use a savekey storage area to store key record values during program execution. You must define savekey storage if your program must do any of the following:

- Update records with the U(pdate) and D(elete) program functions
- Obtain records sequentially with the N(ext) program function
- Display repeated record blocks that the end user can scroll with the F(orward) and B(ackward) functions

Define savekey storage in screen definition or Commarea

You can store savekey data either in:

- The program Commarea, a storage area that Online Express automatically creates when you indicate so on the Savekey Definition screen.
- Fields that you define on the program screen.

If you use screen fields to store the savekey data, you define either one or two savekey fields, depending on your screen design as follows:

- If your screen displays only one occurrence of data at a time, and the data is updateable, you define one savekey field.
- If your screen displays a scrollable or updateable repeated record block as well as a single occurrence of data, you define either:
  - Two savekey fields: one for the repeated block data and one for the single occurrence data.
  - Or one savekey field for both.

Size of savekey is automatically calculated

Online Express automatically calculates the minimum size requirement for savekey storage, and displays that size on the Savekey Definition screen. Your savekey size is the total of the key lengths of each updateable record on your screen, plus a one-byte flag per key.

Commarea also required to store data you pass

Another purpose of a Commarea is to store data that your program passes between programs. If you use the X(CTL), M(SG-SW), or C(all) functions to pass data between programs, you must specify on the Savekey Definition screen the size of the largest record that you must pass. Online Express adds this number of bytes to your Commarea.

Define a Commarea

You define a program Commarea simply by specifying its size on the Savekey Definition screen. Its size should be the number of bytes of the largest record that you pass between programs. If you also use Commarea to store your savekey data, Online Express adds its byte requirements to Commarea if you specify so.

When you define a Commarea to store savekey data, or data that you pass between programs, or both, Online Express creates the following storage area in Commarea when you generate the program:

```
-KYWD-  12-*----20---*----30---*----40---*----50---*--
 SYM2   $PX-CA-COMPUTE-LEN( savekeybytes, sharedbytes)
 CA05   FILLER
        $PX-CA-SETUP
```

where:

- *Savekeybytes* is the number of savekey storage bytes that Online Express calculates for storing record keys when:
  - Obtaining records sequentially with the N(ext) program function
  - Displaying repeated record blocks that the end user can scroll with the F(orward) and B(ackward) functions
  - Re-reading repeated record blocks so that the end user can update and delete them with the M(odify) and E(rase) functions.

    Note: Online Express calculates the number of *savekeybytes* as the key length of each updateable record on the screen, plus a one-byte flag per key. Note the following exception:
  - For SQL repeated record blocks that are scrollable, Online Express calculates *savekeybytes* as the length of each Order By column, plus a one-byte flag per key.
- *Sharedbytes* is the number of bytes that you specify for storing data that you pass between programs with the X(CTL), M(SG-SW), or C(all) functions. The number that you specify can be any number of bytes that you want to pass. Online Express stores the *sharedbytes* in the data structure in Commarea named PX-USER-COMMAREA.

Defining Savekey Storage and a Commarea

To define a savekey storage area and a Commarea, follow these steps:

1 Ensure that you have defined all program functions and database calls for the program.

View storage requirements

2 Display the Savekey Definition screen by selecting option 7, Savekey Definition, from the Online Express menu. The screen displays the savekey storage requirements, in number of bytes, as shown in *Savekey Definition Screen*. If you want to define savekey fields, write down these requirements so you will know how large to define your savekey field(s).

Figure 6-48. Savekey Definition Screen

```
COMMAND ===> _

                                          Savekey
                          Screen Field    Required Bytes
                          ----------------  --------------
FOR NON-REPEATED
            RECORDS ===> SAVEKEY-1        00008

FOR REPEATED RECORDS ===> SAVEKEY-2       00015 * 05 OCCURRENCES = 00075 TOTA
         (BLOCK 1)

OR FOR ALL RECORDS   ===>                 00083

                    ------------ OR -----------

USE COMMAREA FOR SAVEKEY STORAGE REQUIREMENTS ===> N  (Y or N)
         ADDITIONAL COMMAREA BYTES REQUESTED ===> 00000
```

Define savekey storage in screen fields

3 To store savekey data in screen fields, first determine whether to define one or two savekey fields, as follows:

- If your screen updates only one occurrence of data at a time, or your program must execute the N(ext) function, define just one savekey field.
- If your screen displays an updateable repeated record block as well as a single occurrence of data, define either:
  - Two savekey fields: one for the repeated block and one for the single occurrence.
  - Or one savekey field for both.

4 Define your savekey field(s) in your screen definition, using the APS Screen Painter. Set their Length and Type attributes as follows:

Attribute	Setting
Length	The number of bytes specified in the Savekey Required Bytes field on the Savekey Definition screen
Type	P(rotected)

5 Redisplay the Savekey Definition screen and enter the savekey field name(s) in the Screen Field field.

Define savekey storage in the Commarea

6 To store savekey data in the program Commarea, enter y(es) in the field, Use Commarea for Savekey Storage Requirements.

7 To define storage in the Commarea for receiving data that another program passes, enter a number of bytes in the field, Additional Commarea Bytes Requested.

Special Considerations

Define a global Commarea

- To ensure that your Commarea can accommodate both the program key and the largest amount of data that the program can receive from another program, you can define a global Commarea size that all programs of the application use as their Commarea size. To do so, follow these steps:

  a Check the Savekey Definition screen of each program to determine the largest savekey requirement.

  b Determine the number of bytes of the largest record that your program can receive.

  c Add these two numbers.

  d In a control file or a USERMACS macro, assign the result to the APS variable named &TP-USER-LEN. Online Express uses the value of this variable to assign the size of the savekey storage in Commarea. The format of &TP-USER-LEN is as follows.

  ```
  &TP-USER-LEN = savekeybytes + sharedbytes
  ```

  where:

  - *Savekeybytes* is the value of the Savekey Required Bytes for All Records field.

- *Sharedbytes* is the value of the Additional Commarea Bytes Requested field.

Suppress generation of savekey storage area

- You might want to suppress the generation of the savekey storage area and savekey logic for a call if you want to:
  - Update or delete records that do not have a unique index
  - Perform blind update or delete functions

  To do so, after you define the call, display the Database Access Summary screen and enter either *update (for a Modify call) or *delete (for an Erase call) next to the call in the Function field. For example, to update and delete a record, you would write two separate calls--one to obtain and modify the record and one to obtain and delete the record. You would enter the following values for each call:

Call	Function Field Value	Action Field Value
01	*update	OM
02	*delete	OE

7 Generate the Application

This chapter contains the following sections:

- *Concepts of Generation*
- *Setting Options*
- *Generating Applications*
- *Executing Applications*

Concepts of Generation

You can generate an entire application or you can generate selected programs and screens of an application. When you generate an application, the APS Generator and APS Precompiler translate APS specifications into a complete structured COBOL application. APS then passes the source to your COBOL compiler and link edit program to produce a load module.

Tailor generation

You tailor how APS generates an application using options and job submission modes. Options are available for controlling both the Generator and Precompiler, as well as target specific options.

When APS generates an application, it:

- Ensures that each component of the application exists.
- Generates screen symbols for each screen for use by the Precompiler.
- Generates screen source for use by the DC environment.
- Rearranges the specifications programs into proper COBOL program organization.

- Includes externally-defined information that the program references, such as copylibs and user-defined macros at the appropriate COBOL program locations.
- Processes all database and data communications calls and user-defined macros, translating all source to COBOL source.
- Translates all APS Report Writer source to COBOL source.
- Writes a temporary error message file and merges it with the COBOL compiler error message file. The combined error message file presents messages sorted by program line number with both types of messages appearing where appropriate.

APS stores generated and precompiled COBOL program source and screen output in the following data sets in your user Project and Group, depending on the DC target specified:

DC Target	Generated Screen, Mapset Output File	Generated Program Output File
CICS	GENBMS	COBCIC
IMS	GENMFS	COBIMS
ISPF Dialog	GENDLG GEN5DLG (Mod 5)	COBDLG
ISPF prototype	GENPANEL GEN5PANL(Mod 5)	COBISPF
MVS (batch)	Not applicable	COBMVS

Setting Options

Define development environment

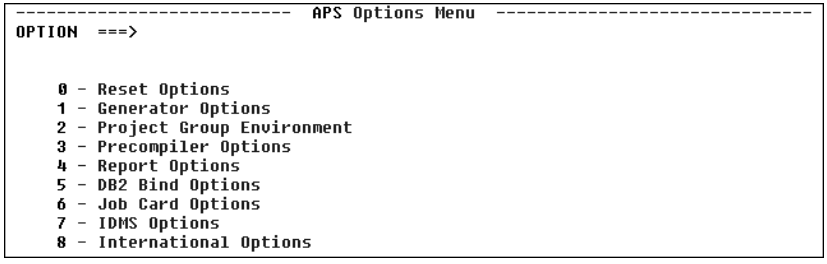
Before you generate an application, you must set options to define the development environment appropriately. You set options for:

- Project and Group
- APS Generator
- APS Precompiler
- IDMS
- SQL Bind and Translate

APS sets option default values for these options according to your installation configuration.

To access the APS Options menu, from the APS Main Menu enter option 0 in the Command field. Alternatively, from any APS screen, enter opt in the Command field.

Figure 7-1. APS Options Menu



Setting Project and Group Options

Specify to APS the Project and Group location of your application and where you want APS to generate the Project and Group DDIFILE data set. If you use the APS Data Element Facility or the APS/ENDEVOR interface, specify their locations as well. To do so, follow these steps:

1 Access the Project Group Environment screen. To do so, from the APS Options Menu, enter option 2 in the Option field. Alternatively, from any APS screen enter opt 2 in the Command field. The Project Group Environment screen displays.

2 Complete the fields on the Project Group Environment screen as follows:

Field	Description
Project	The name of the Project. For example, myproj. Must be 1-8 alphanumeric characters; the first character must be alphabetic.
Group	The name of the Group. For example, mygrp. Must be 1-8 alphanumeric characters; the first character must be alphabetic.

Field	Description
DDIFILE	The location of the Project and Group's DDIFILE data set; do not specify the name DDIFILE. Default: The Project and Group path specified above. For example, *myproj.group*.
Data Element Library Prefix	Optional. The location of the Data Element Facility APSDE data set; do not specify the name APSDE. For example, apspg.project1.group1. For information on the Data Element Facility, see Administrator's Guide: Chapter 2, "Managing Data Elements."

Setting Generator Options

Set the APS Generator options appropriately for your environment. To do so, follow these steps:

1 Access the Generator Options screen. To do so, from the APS Options Menu enter option 2 in the Option field. Alternatively, from any APS screen enter opt 2 in the Command field. The Generator Options screen displays.

Figure 7-2. Generator Options Screen

```
--------------------------  APS Generator Options  ---------------------------
COMMAND ===>

 TARGET OS ===>        (MVS, VSE)

        DC ===>        (IMS, CICS, DLG, MVS, or ISPF(prototyper))
        DB ===>        (IMS, DLI, VSAM, SQL, OR IDMS)
       SQL ===>        (Blank, DB2, SQLDS)
                              JOB DEST ===>
 JOB CLASS ===>            CARDIN MEMBER ===>
 MSG CLASS ===>
                       GENERATE COBOL-II ===>       (Yes or No)
   LISTGEN ===>          COBOL COMPILER  ===>       (1, 2 or 3)
     COBOL ===>             1 = OS/VS COBOL (GENERATE COBOL-II = NO)
    OBJECT ===>             2 = COBOL-II
   MFS/BMS ===>             3 = COBOL for MVS
    GENSRC ===>
 APS DEBUG ===>
 USER HELP ===>             CICS RELEASE ===>       (Blank, A or B)
                             IMS RELEASE ===>       (Blank, A or B)
                                   SUPRA ===>       (Yes or No)

  APS Parm ===>
COBOL Parm ===>
```

2 Set options appropriate for your environment as described below.

Option	Description and Values	
Target OS	Operating system.	
DC	Data communications target. For valid DB/DC combinations see the "DB/DC Target Combinations" topic in the APS Reference.	
DB	Database target. For valid DB/DC combinations see the "DB/DC Target Combinations" topic in the APS Reference.	
SQL	SQL target.	
Job Class	Specify any job class valid at your site and known to the APS generators.	
Msg Class	Site-specific.	
Listgen	*Yes*	Generates listing of generated code. See the APS Error Messages manual for a sample.
	No	Default.
COBOL	*Yes*	Saves generated COBOL program source in the library or data set appropriate for your DC target. For the complete list of libraries and data sets.
	No	Default.
Object	*Yes*	Saves generated object code in appropriate library.
	No	Default.
MFS/BMS	*Yes*	Saves generated BMS or MFS mapsets in the GENBMS or GENMFS libraries.
	No	Default.
GENSRC	*Yes*	Saves generated source code in the GENSRC PDS or data set.
	No	Default.
User Help	*Yes*	Enables generation of APS User Help Facility source files.
	No	Default.
Job Dest	Site-specific.	
CARDIN Member	Specify the CNTL library APSDBDC member.	

Option	Description and Values	
Generate COBOL II	*Yes*	Generates COBOL II source code.
	No	Default.
COBOL Compiler	*1*	OS/VS COBOL (Generate COBOL II = *No*)
	2	COBOL II
	3	COBOL for MVS
CICS Release	Specify the CICS release at your site.	
IMS Release	Specify the IMS release at your site.	
SUPRA	*Yes*	Passes SUPRA procedural statements through APS unchanged.
	No	Processes SUPRA procedural statements.
APS Parm	Overrides the APS Parm field on the Precompiler Options screen. Displays all options whose default values you have overridden in the Precompiler Options screen. You can temporarily override these values simply by overtyping them in this field, but changes made here are not saved; they remain in effect only until you exit APS.	
COBOL Parm	Specify parameters or directives for COBOL compiler. See the ***COBOL Language Operating Guide*** for valid values.	

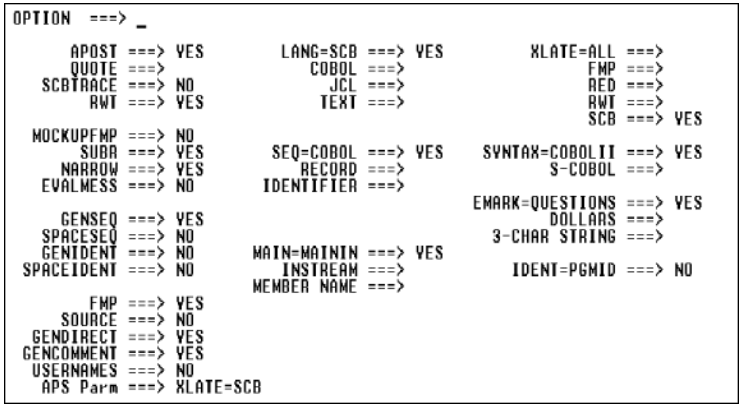
Setting Precompiler Options

Set the APS Precompiler options appropriately for your requirements or preferences. To do so, follow these steps:

Access the Precompiler Options screen

1 Access the Precompiler Options screen. To do so, from the APS Options Menu enter option 3 in the Option field. Alternatively, from any APS screen enter opt 3 in the Command field. The Precompiler Options screen displays.

Figure 7-3. APS Precompiler Options Screen



2 Set options appropriate for your environment as described below.

Option	Description and Values	
Apost	Overrides Quote.	
	Yes	Default. Lets you use the apostrophe character to delimit non-numeric literals in your input source.
Quote	Overrides Apost.	
	Yes	Lets you use the single quote character to delimit non-numeric literals in your input source.
	No	Default.
SCBtrace	*Yes*	Activates the SAGE-TRACE-FLAG debugging facility.
RWT	*Yes*	Default. Generates COBOL code from APS Report Writer statements. Specify with COBOL II compiler.
	No	Passes Report Writer statements directly to the COBOL compiler.

Option	Description and Values	
		Note: For very large Report Writer programs, enter rwt=bigrwt in the APS Parm field on the Generator Options screen.
Lang	Indicates which type of source to process and which columns to process.	
	SCB=yes	Default. Processes APS specifications (S-COBOL) in columns 8-72; the symbol &07 in your code forces a character into column 7.
	COBOL=yes	Processes COBOL source in columns 1-72.
	JCL=yes	Processes JCL in columns 1-72. Useful for text-processing JCL and for controlling columns 1-6 of S-COBOL
	Text=yes	Processes any source in columns 1-80. All columns are considered text; no sequence numbers are generated. Automatically sets XLATE=FMP. To override XLATE=FMP, enter XLATE=*value* in the APS Parm field.
Evalmess	*Yes*	Generates messages that list evaluation bracket resolutions. Usually results in long listings.
	No	Default.
Seq	Specifies the type of sequence numbers that APS generates. See also, Genident, Spaceident, Ident.	
	COBOL=yes	Generates COBOL-style numbers in columns 1-6.

Option	Description and Values	
	Record=yes	Generates new numbers in columns 73-80, incrementing by 100 for each input record and by two for each generated record.
	Identifier=yes	Generates line numbers in columns 73-80; columns 73-74 contain 0.
Syntax	Specifies which compiler to use.	
	COBOLII=yes	Generate COBOL-II syntax.
	S-COBOL=yes	Generate S-COBOL syntax.
Emark	Generates a three-character string marking error and warning messages in the message report.	
	Questions=yes	Default. Generates ???.
	Dollars=yes	Generates $$$.
	3-Char String= string	Generates the string you specify.
Genseq	Overrides Spaceseq.	
	Yes	Default. Generates sequence numbers in columns 1-6 for blank or out-of-sequence lines of source code and when new lines are generated.
Spaceseq	Overrides Genseq.	
	Yes	Generates spaces in columns 1-6; incompatible with Lang=Text.
Genident	See also, Spaceident, Ident, Seq.	
	Yes	Generates sequence numbers in columns 73-80 for blank or out of sequence source code lines and when new lines are generated.

Option	Description and Values	
	No	Default. Generates the last known contents of columns 73-80 when new lines are generated and passes identifiers as they exist in GENSRC.
Spaceident	See also, Genident, Ident, Seq.	
	Yes	Generates spaces in columns 73-80. Incompatible with Lang=Text.
Main	Specifies location of the main input source.	
	MAININ=yes	Default. Reads from file named by external name MAININ. Use this default unless using your own JCL.
	Instream=yes	Reads source instream with the JCL that you provide.
	Member Name= membername	Reads from the PDS or file name or source statement library designated by the external name SCELIB.
Ident	See also, Genident, Spaceident, Seq.	
	Yes	Generates the internal program name in columns 73-80.
	No	Default.
FMP	*Yes*	Default. Processes APS macros and user-defined Customization Facility macros.
	No	Use only with your own JCL skeleton.
Source	*Yes*	Prints the main input source program, specified in the MAIN option, after the message report.
	No	Default.

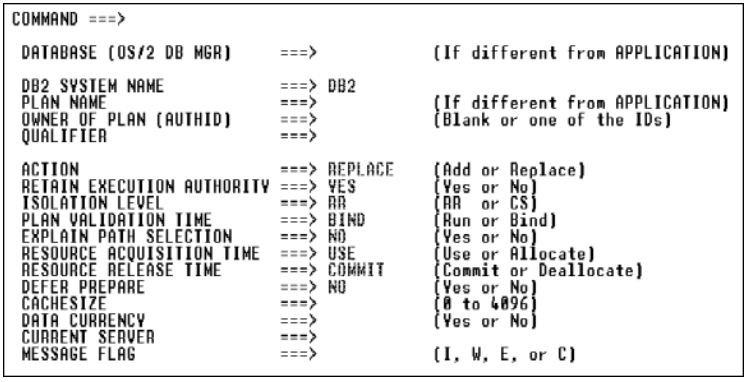
Option	Description and Values	
Gendirect	*Yes*	Allows generatation of nested IF statements in the COBOL source.
Gencomment	*Yes*	Generates replaced source statements as comments in the COBOL source.
	No	Default.
Usernames	*Yes*	Generates the following prefix for APS-generated paragraphs: *paraname-*
	No	Default. Generates the following prefix for APS generated paragraphs: G--
		Note: To generate any other prefix, enter the following in the APS Parm field on this screen: *usernames=prefix*
APS Parm	Displays all Precompiler options whose default values you override. These values also display in the APS Parm field on the Generator Options screen. APS saves the values you change on the APS Parm field on the Precompiler Option screen. APS does not save values that you change in the APS Parm field on the Generator Options screen.	

Setting SQL Bind and Translate Options

Specify Bind and Translate options. To do so, follow these steps:

1 Access the SQL Bind and Translate Options screen. To do so, from the APS Options Menu enter option 5 in the Command field. Alternatively, from any APS screen enter opt 5 in the Command or Option field. The APS Bind Options screen displays.

Figure 7-4. DB2 Bind Options Screen



2 Select Bind and translate options appropriate for your environment as described below.

Field	Description and Values
DB2 System Name	Specify the appropriate name for your site. Default: DB2.
Plan Name	Specify the plan name you use when you Bind an application. If you leave this field blank, the default depends upon your use of the BIND command in the Application Painter.
Owner of Plan (Authid)	Leave this field blank or specify a primary or secondary authorization ID of the BIND.
Qualifier	Leave this field blank or specify the implicit qualifier for the unqualified table names, views, indexes, and aliases contained in the plan.
Action	Specify the bind action to be executed. Valid values: *add* or *replace*.
Retain Execution Authority	Specify *Yes* if you specified REPLACE in the BIND ACTION field. Otherwise specify *No*.
Isolation Level	Valid values: *rr* or *cs*.
Plan Validation Time	Valid values: *run* or *bind*.

Field	Description and Values
Explain Path Selection	*Yes* Activates the DB2 EXPLAIN function. *No* Does not activate the function.
Resource Acquisition Time	Valid values: *use* or *allocate*. If you enter ALLOCATE, you must enter DEALLOCATE in the Resource Release Time field.
Resource Release Time	Valid values: *commit* or *deallocate*. The value you enter in this field depends on the value you entered in the Resource Acquisition Time field.
Defer Prepare	*Yes* Generates the keyword DEFER(PREPARE), which defers the prepare statement referring to a remote object. *No* Default.
Cache Size	Specify the size (in bytes) of the authorization cache to be acquired in the EDMPOOL for the plan. Valid values: 0 to 4096.
Data Currency	*Yes* Data currency is required for ambiguous cursors. *No* Data currency is not required for ambiguous cursors.
Current Server	Leave this field blank or specify a connection to a location before the plan runs.
Message Flag	Specify which messages display. Valid values: *I, W, E, C,* or blank.

Setting Job Control Cards

You can create up to five job cards - named J1 through J5 - with varying job names, account information, classes, and other attributes. To do so, follow these steps:

1 Access the Job Control Cards screen. To do so, from the APS Options Menu enter option 6 in the Option field. Alternatively, from any APS screen enter opt 6 in the Command or Option field. The Job Control Cards screen displays.

2 Modify the cards as desired.

Setting IDMS Options

Specify IDMS options as follows:

1 Access the IDMS Options screen. To do so, from the APS Options Menu enter option 7 in the Option field. Alternatively, from any APS screen enter opt 7 in the Command or Option field. The IDMS Options screen displays.

2 Specify IDMS options appropriate for your environment as described below.

Option	Description and Values	
Dictionary Name	Specify the dictionary name.	
Central Version or Local	Specify the compile environment. APS generates a SYSTRNL with a unique DSN whose high level qualifier is your user ID.	
	cv	Default. Central Version.
	local	When you specify local, also enter a volume in the IDMS Local Jrnl Disk Vol field.
	dummy	When you specify dummy, APS generates a SYSTRNL DD DUMMY
IDMS Local Jrnl Disk Vol	Local compile disk volume for journal.	

Option	Description and Values	
Dictionary Update	*Yes*	Log program compile information to the dictionary.
	No	Default. Do not log program compile information.
IDMS DMLC Output to PDS	*Yes*	Write DMLC compile statements to a PDS. If you enter yes, you must allocate a &DSN..IDMSOUT PDS prior to compilation.
	No	Default. Do not write DMLC compile statements to a PDS.
IDMS Loadlib Qualifier	Specify full qualifiers for IDMS..LOADLIB.	
IDMS SYSCTL DSN	Optional. Specify DSN of IDMS dictionary.	
CV Node Name	Specify name of central version DDS (Distributed Database System) node under which loadlib program is compiled.	
DMLIST (List Generation)	*Yes*	Generate list.
	No	Default.
Generate DB-BIND in Pgm	*Yes*	Do not suppress the generation of the DB-BIND macro.
	No	Suppress the generation of the DB-BIND macro. You must manually code the DB-BIND macro in your program.
IDMS Password	N/A	
IDMS 12.0 SYSIDMS DSN	Specify the name of the IDMS 12.0 dataset.	
Include IDMSLIB	Specify the appropriate dataset name for CICS, MVS or other environments.	

Resetting Profile Variables

You can reset the profile variables of a Project and Group to their original installation values. For information on original installation values, see the ***Installation Guide*** chapter *Installing APS for zIOS*.

This option automatically resets all of the following types of profile variables:

- All APS Profiles Variables
- All APS library prefixes and DSNs
- Generator Options screen options
- IDMS Options screen options
- Job Control Cards screen options
- Precompiler Options screen options
- DB2 Bind options

Reset all of the above options as follows:

1. Access the APS Options menu. To do so, from the APS Main Menu enter option 0 in the Command field. Alternatively, from any APS screen, enter opt in the Command or Option field. The APS Options menu displays.
2. Select option 0. APS immediately resets the options and displays a message informing you that the profile pool has been reset.

Generating Applications

You can generate your entire application all at once or you can generate selected programs and screens individually. To do so, follow these steps:

Ensure that your last session ended normally

1. Ensure that you exited your previous APS session normally; if you exited abnormally and then submit a generation job, the job will fail. In this case, exit APS normally, re-start APS, and resubmit the job.

Set generation options

2 Ensure that your generation options are set appropriately, as described in *Setting Generator Options*.

3 Display the Application Painter and enter ap in the Type field and the application name in the Member field.

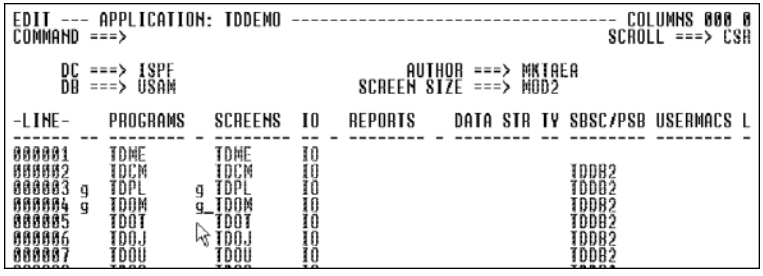
Generate application

4 To generate your entire application, enter gen in the Command field.

Generate programs or screens individually

5 Alternatively, to generate one or more programs or screens individually, enter g next to those program or screen names, as shown in *Generating Programs and Screens Individually*. To generate all screens, enter *generate sc all* in the Command field, or enter g next to all screens; to generate all programs, enter generate pg all in the Command field, or enter g next to all programs.

Figure 7-5. Generating Programs and Screens Individually



Check job results

6 Check the result of the jobs in SDSF by selecting Services Job Queue.

Special Considerations

Override BMS mapset names

- In addition to generating APS screen symbols, APS generates a BMS mapset for each CICS screen, and assigns a default name to each mapset. To override a BMS mapset name, see *Paint Character Screens*.

- ***Generate BMS multiple-map mapsets in APS***
  To generate a BMS multiple-map mapset that includes some or all screens of your application, do one of the following:

  - To include all screens in a multiple-map mapset, enter gen ms mapsetname in the Command field. This name overrides each screen's default mapset name, which is displayed on the Screen Generation Parameters screen. For more information on default mapset names, see *Setting Parameters for Generation*.
  - To include selected screens in a multiple-map mapset, use the APS BMS Multiple-Map Mapset screen to specify the screens and generate the mapset. To display this screen, select option 4, Utilities from the APS Main Menu, and then select option 1, Non-Painted APSSRC/GENSRC Compilation. The APS Precompiler screen displays. On it, select option 3, Generate BMS Multiple-Map Mapset.

  Important: After you generate a multiple-map mapset using APS, you must compile, link, and generate the BMS source in your CICS environment.

Executing Applications

Run from APS

The APS Prototype Execution facility allows you to execute and test applications. This facility provides 2 environments (IMS and CICS)in which you can debug and execute an application and test all data communication and database functions.

Execute CICS and IMS applications

You can execute CICS and IMS DC applications using the APS Prototype Execution facility, which emulates the basic functions of the mainframe CICS and IMS environments. To use this facility, you must specify the DC target ISPF when you generate your application. Using this facility, you can test all data communication and database functions and access all database environments except IMS.

Use the following execution facilities to execute fully functional applications that access your databases:

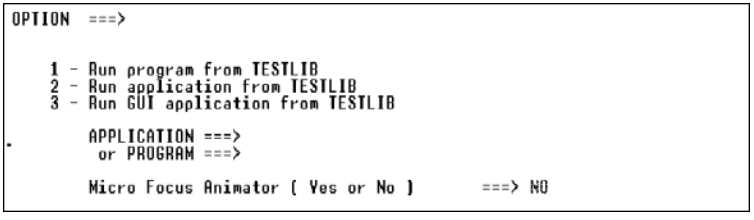
Application	APS for z/OS Execution Facility
CICS	APS Prototype Execution (DC target = ISPF)
IMS DC	APS Prototype Execution (DC target = ISPF)
MVS (batch)	N/A
ISPF Dialog	N/A

To execute and test your application using the APS Prototype Execution Facility, follow these steps:

Access the execution facilities

1 From the APS Main Menu, enter option 3 in the Command field. The APS Prototype Execution screen displays.

Figure 7-6. APS Prototype Execution Screen



2 Select the appropriate option to display your program or application, and execute it.

8 Create User Help

This chapter contains the following sections:

User Help Facility Concepts

Intergrate help logic

The APS User Help Facility allows you to integrate logic into character applications to display help information. To implement user help, compile and generate the APS provided help application and recompile your user application. Three programs comprise the user help application, APSUHELP, one of which becomes part of your user application. If desired, you can customize the help application to conform to programming conventions at your site. To customize APSUHELP, you change the default values of the variables stored in the APHLPIN control file. For detailed information regarding customization, see *Customizing the User Help Application*.

Transfer control to help application

When a user requests help, the application program transfers control to the help application program responsible for displaying help information. APS saves the current user application screen in the help database or in temporary storage. When your user application program transfers control to the help display program, it passes information regarding the type of help information requested. The help display program reads the help database and displays the information to the user. When control is returned to your application

program, the help database or temporary storage area is read again to restore the current screen.

Since the help display program is part of your user application, you must compile it in a manner that is consistent with the other programs in your application. That is, you compile it for the appropriate DC target and if your applications use a COMMAREA or a SPA, then the help display program must also have a COMMAREA or SPA of the same length.

To create help for your applications, you:

- Define the help database.
- Create the help source file.
- Generate the APSUHELP application.
- Load the help database.

Create four types of user help

The APS User Help Facility lets you create user help for your user applications. With this facility, you can create the following types of help:

- Application help describes the user application and its main screen options.
- Screen help describes a screen and its options.
- Field help describes the field where the cursor is positioned.
- Field value help displays a list of valid values that end users can select.

You create a help source file to store the help text for one or more user applications. You load the help source file into the help database. Once the help source file is loaded into the help database, you or your end users can add, modify, or delete any help text in the help database.

Defining the Help Database

Define help for multiple DB targets

You can define help databases for IMS, VSAM, and SQL. Each database is described below. If required, you can change the default help database

names to conform to your site's naming conventions. For more information, see *Customizing the User Help Application*.

IMS	A two-level database. The access method used is HDAM. The default database name is HELPDBD. Its parent segment name is HELPSEG and its child segment name LINESEG.
VSAM	A KSDS variable length file. Its maximum length is 3771 bytes: its minimum length is 121 bytes. The default database name is HELPVSM.
SQL	One variable length table. Refer to the SQL description in the SQLDDL datasets, HELPDB2 (DB2), for more details.

The primary key for these databases is 42 bytes. It is structured as follows:

Byte	Value
1	Entity Type where: A=Application S=Screen D=Field V=Field value
2-39	Application name Screen name Screen + field name
40-42	000 or context number (global fields)

Defining an IMS Help Database

Define an IMS database for user help, as follows:

1. Generate the help database description (DBDGEN) for HELPDBD. To do so, enter 2 in the Command field. From the Dictionary Services screen, enter 1 in the Command field. From the Importer Facilities screen, enter 2 in the Command field.
2. On the IMS screen, type the DBDSRC member name, helpdbd, in the Member field and enter 1 in the Command field. Note: You must create JCL to define the VSAM space for the help database specific to your site.

3 Generate the help program specification blocks (PSBGEN). There are two PSBSRC members, HELPPSBL and HELPPSB. To do so, enter 2 in the Command field. From the Dictionary Services screen, enter 1 in the Command field. From the Importer Facilities screen, enter 2 in the Command field

4 On the IMS screen, type the PSB member name, in the Member field and enter 2 in the Command field.

5 Zeroload or initialize the help database.

6 Optionally, regenerate DDI symbols. To determine if you must perform this step, see *Special Considerations*.

Defining a VSAM Help Database

Define a VSAM database for user help as follows:

1 Generate IDCAMS control statements. To do so, enter option 2, Dictionary Services in the Command field on the APS Main Menu. Enter option 1, Import Facilities in the Command field on the Dictionary Services screen. Enter option 3, VSAM on the Import Facilities screen.

2 Type option 2, Generate IDCAMS (VSAM) Input into AMSERV on the VSAM Importer screen and type helpvsm in the Member field and press Enter.

3 Zeroload the help database.

4 Optionally, regenerate DDI symbols. To determine if you must perform this step, see *Special Considerations*. To regenerate DDI symbols, enter option 2, Dictionary Services in the Command field on the APS Main Menu. Enter option 1, Import Facilities in the Command field on the Dictionary Services screen. Enter option 3, VSAM on the Import Facilities screen. Type option 3, Generate DDISYMB Symbols from DDIFILE on the VSAM Importer screen and type helpvsm in the Member field and press Enter.

Defining SQL Help Databases

Define an SQL database for user help as follows:

1 Create the help database using the SQL statements in the SQLDDL dataset, HELPDB2 (HELPDB2). For example:

```
CREATE TABLE HELPXDB
 (H_PRIME_KEY                  CHAR (42)  NOT NULL,
 (H_BUSINESS_NAME              CHAR (55),
 (H_CONTEXT_NAME               CHAR (8),
 (H_LST_UPD_DATE               DECIMAL (7) NOT NULL,
 (H_LST_UPD_TIME               DECIMAL (9) NOT NULL,
 (H_LINE_COUNT                 DECIMAL (3) NOT NULL,
 (H_LINE_TBL_AREA              VARCHAR(3802));
```

2 To optimize performance, create an index on column H_PRIME_KEY.

3 Set the Target option on the APS Generator Options screen to specify the SQL database target. Valid options are SQLDS, DB2 and SQL400.

4 Optionally, regenerate DDI symbols. To determine if you must perform this step, see "Special Considerations" below.

Special Considerations

- If your application database is the same type as your help database, it is not necessary to regenerate the user application program's DDI symbols. However, if the database types are different, you must include the description for the help database in your DDI input and regenerate.
- If you must generate DDI symbols, ensure that variable &HELP-SUBSCHEMA-ADD=no in the APHLPIN control file. If you do not, then &HELP-SUBSCHEMA-ADD =yes. Use the table below to help you determine when you must regenerate DDI symbols.

Application Database	Help Database	Regenerate DDI
VSAM	VSAM	No
VSAM	SQL	No
VSAM	DLI	No
VSAM/Other	VSAM	Yes

Application Database	Help Database	Regenerate DDI
VSAM/Other	SQL	No
VSAM/Other	DLI	Yes
IMS	IMS	No
IMS	SQL	No
IMS/Other	IMS	Yes
IMS/Other	SQL	No
SQL	SQL	No
SQL	VSAM	Yes
SQL	DLI	No
SQL/Other	SQL	No
SQL/Other	VSAM	Yes
SQL/Other	DLI	Yes

Working with the Help Source File

Use any of the following User Help source utilities to create your help source file. Before executing these utilities, ensure that your help database has been defined and created and that the help application, APSUHELP, has been generated. In addition, before you create the help source, ensure that your user application and screens have been created.

Applications Utility	Lets you create a complete help application in one session. If your user application contains global data elements, you must also use the Data Elements Utility.
Screens Utility	Lets you create help for individual screens, as well as field help, field value selection lists, and messages.
Data Elements Utility	Lets you create help for global fields that reside in the APS Data Element Facility.

The help source file you create is an ASCII text file. The help source file layout is as follows:

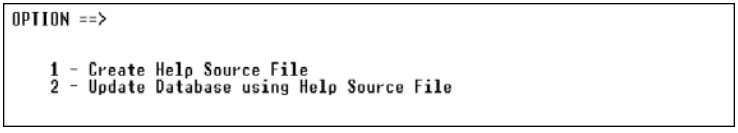
Byte	Value	Description
1-3	001 002 003	First header record Second header record Text records (optional)
4	D V A S	Field description Field value Application description Screen description
5-42	*screenname* + *fieldname* *application name* *screenname*	
43-45	000 001	Text description Text value
First header record		
46-53 54-59 60-67 68-121	Context name Date created Time created Blank	
Second header record		
46-100 101-121	Business name Blank	A descriptive name that easily identifies the user application and its components.
Text record		
46-48 49-121	Numeric counter *description* or *value*	

Creating the Help Source File

To create the help source file, follow the steps below.

1 From the APS Main Menu, enter option 2 in the Command field. Then enter option 6 in the Command field. APS displays the User Help Facility screen.

Figure 8-1. User Help Facility Screen



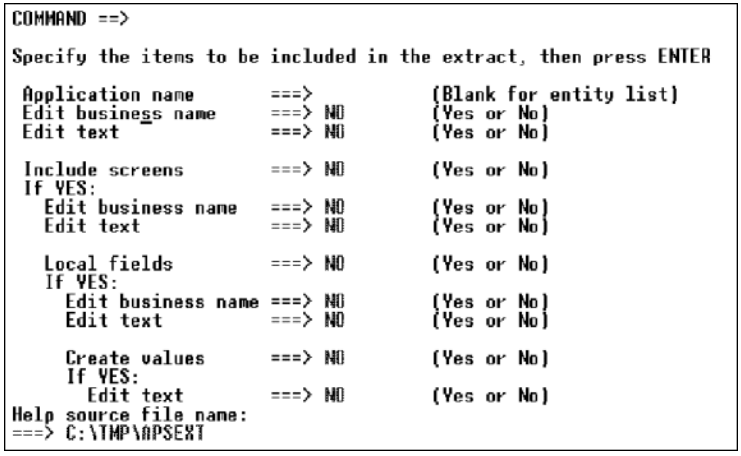
2 From the User Help Facility screen, enter 1 in the Command field. APS displays the User Help Source Utility screen.

Figure 8-2. User Help Source Utility Screen

3 From the User Help Source Utility screen, select a utility to create your help source file.

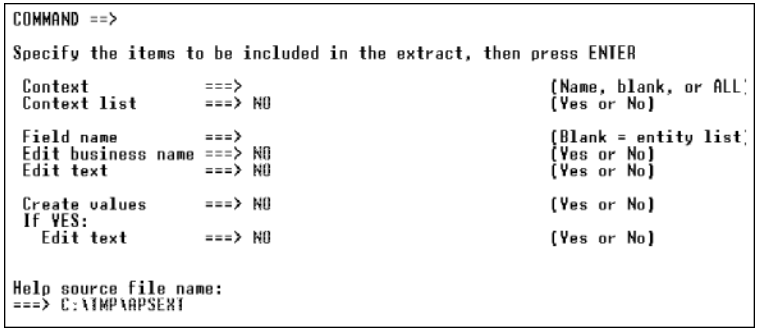
- If you select , Applications, APS displays the Applications Utility screen.

Figure 8-3. Applications Utility Screen



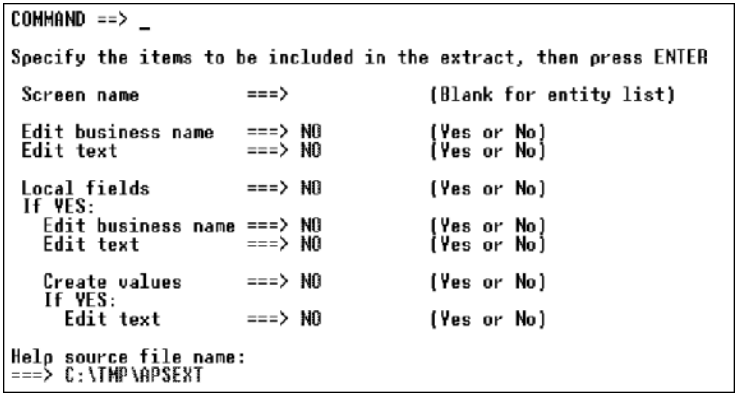
- If you select Data Elements, APS displays the Data Elements Utility screen.

Figure 8-4. APS User Help Data Elements Utility Screen



- If you select Screens, APS displays the Screens Utility screen.

Figure 8-5. APS User Help Screens Utility Screen



4 Complete the fields for the utility selected as follows:

Field	Screen	Description
Context Name	Data Elements	Type the context name associated with the field to display all the fields with that context. Leave this field blank to display the fields with no context. Type all to display all the fields with their contexts. Select a name from the selection list by entering s next to it.
Context List	Data Elements	*No* Do not create a context list. *Yes* Create a context list.
Application Name	Applications	Enter the user application name, or leave this field blank and press Enter to display a selection list. Select a name from the selection list by entering s next to it.

Field	Screen	Description
Field Name	Data Elements	Enter the field name or leave this field blank and press Enter to display a selection list. Select a name from the selection list by entering s next to it.
Screen Name	Screens	Enter the screen name, or leave this field blank and press Enter to display a selection list. Select a name from the selection list by entering s next to it.
Edit Business Name	All	A business name is a descriptive name that easily identifies the user application and its components.
		No Business name defaults to the user application name.
		Yes Assign a business name.
Edit text	All	*No* Do not create help text.
		Yes Create help text.
Include Screens	Applications	*No* Do not create screen help.
		Yes Create screen help.
Include Fields	Applications	*No* Do not create field help.
		Yes Create field help.
Local Fields	Applications and Screens	*No* Do not create local field help.
		Yes Create field help.
Create Values	All	*No* Do not create field value help.
		Yes Create field value help.
Help Source File Name	All	Help source filename, APSEXT. If this file already exists, it is overlaid..

5 After completing all fields for the user application components that you want to create help for, press Enter. The APS User Help Facility extracts the names of the user application screens and local fields to create the help source file.

6 If you entered Yes in the Edit Business Name field, APS displays the Edit Business Name screen. Enter a business name with a maximum of 55 characters and press PF3.

Create help text

7 If you entered Yes in the Edit text field, APS displays the Help Text Edit screen. You can enter up to 50 lines of help text, 73 characters per line. Edit text using ISPF line commands. If you are creating a field value selection list, enter one value per line. If your end users will create the help text, press PF3 to leave this screen blank.

Figure 8-6. Edit Business Name Screen

```
COMMAND ==>

Specify the business name, then press END

 Description for  APPLICATION    LWMAPPL

 Context is

 Business name ===> Test Application_
```

Figure 8-7. Help Text Edit Screen

```
COMMAND ==>

Specify the business name, then press END

 Description for  APPLICATION    LWMAPPL

 Context is

 Business name ===> TEST APPLICATION


 USER HELP TEXT EDIT FOR APPLICATION   LWMAPPL              COLUMNS 001 072
 COMMAND ===>                                              SCROLL ===> PAG
 ****** ********************************* Top of Data ********************************
 ******
 ******  This menu allows you to select:
 ******
 ******  1   To create a new file for a customer and update the database
 ******  2   To update an existing file and update the database
 ******  3   To delete and file and update the database
 ******
```

Special Consideration

APS converts text entered on the Help Text Edit window to upper case when you save or press Enter.

Generating the User Help Application

Verify APSUHELP application

Before you generate help for your user application, ensure that your project.group data sets contain the APSUHELP application software described below.

Data set	Module	Description
APSAPPL	APSUHELP	APSUHELP application
APSPROG	A1UHUPD APSDISP APSFM	APSUHELP application programs
APSSCRN	APDI APSFM	APSDISP program screen APSFM program screen
APSREPT	A1UHUPD	Report layout for load program
APSDATA	HMCOMM	Data structure used by APSFM
COPYLIB		Copylib members for:
	HELPCOPY HELPDB2 HELPROOT HELPLINE A1UHFILE	VSAM help database DB2 help database IMS help database IMS help database Batch load program
		Note: Ensure that there are no naming conflicts with these copylib members and existing copylib members at your site.
DDISRC	HELPVSM HELPDBD	VSAM database description IMS database description
SQLDDL	HELPDB2	DB2 database description
DBDSRC	HELPDBD	IMS help database description

Data set	Module	Description
PSBSRC	HELPPSBL	IMS PSB for A1UHUPD program
	HELPPSB	IMS PSB for APSDISP and APSFM program

Implement user help

You incorporate the help you create into the user application through an APS-provided application, APSUHELP. This application contains three programs that interface with your user application to make help available. Do not modify these programs. If you modify these programs, you must retrofit your modifications to subsequent releases of this product. The APSUHELP programs are described below.

Program	Type	Description
A1UHUPD	Batch	Loads the help source file into the Help database.
APSDISP	Online	Displays the contents of the help source file. When users request help, control is transferred from your application to this program.
APSFM	Online	Allows you to interactively maintain the help database.

Generating User Help in CICS/ISPF Environments

1. Compile A1UHUPD. Ensure that you have typed yes in the User Help field on the Generator Options screen and that the SUBR option on the APS Precompiler Options screen is set to no. To compile, access the Application Painter screen and type vsam in DB field and ispf in the DC field.
2. Generate the APDI screen. To compile, access the Application Painter screen and type vsam in DB field and ispf or cics in the DC field.
3. Generate the APSFM screen. To compile, access the Application Painter screen and type vsam in DB field and ispf or cics in the DC field.

4 Compile APSDISP. Ensure that you have typed yes in the User Help field on the Generator Options screen. To compile, access the Application Painter screen and type vsam in DB field and ispf or cics in the DC field.

5 Compile APSFM. Ensure that you have typed yes in the User Help field on the Generator Options screen. To compile, access the Application Painter screen and type vsam in DB field and ispf or cics in the DC field.

6 Recompile your application. Ensure that you have typed yes in the User Help field on the Generator Options screen.

Generating User Help in an IMS Environment

1 Compile A1UHUPD. Ensure that you have typed yes in the User Help field on the Generator Options screen and that the SUBR option on the APS Precompiler Options screen is set to no. To compile, access the Application Painter screen and type ims in the DC and DB fields.

2 Generate the APDI screen. To do so, access the Application Painter screen and type ims in the DC and DB fields.

3 Generate the APSFM screen. To compile, access the Application Painter screen and type ims in the DC field and the DB field.

4 Compile APSDISP. Ensure that you have typed yes in the User Help field on the Generator Options screen. This program must be defined to IMS with the PSB, HELPPSB. If your application is conversational, then APSDISP must be conversational as well. Set &TP-USER-LEN the same as your application program.

5 Ensure that each program of your user application for which you want to create help contains a modifiable alternate I/O PCB and that the PCB for the help database is included in each user application program PSB.

6 Compile APSFM. To compile, access the Application Painter screen and type ims in DB field and ims in the DC field.

7 Recompile your application for user help. Ensure that you have typed yes in the User Help field on the Generator Options screen.

Note: User help does not work for user applications that use $TP-SCRNLIST to read multiple screens.

Special Considerations

- When application programs transfer control to APSDISP and returns, your application program is PROGRAM-INVOKED from APSDISP. If you code logic for the PROGRAM-INVOKED paragraph, ensure that control is returned to the appropriate application program.
- In an IMS environment, user help does not work for user applications that use $TP-SCRNLIST to read multiple screens.
- APSDISP must be defined to IMS with the PSB, HELPPSB. If your application is conversational, then APSDISP must be conversational as well. If the size of &TP-USER-LEN varies from program to program, set the size of &TP-USER-LEN to the largest value.

Loading the Help Database

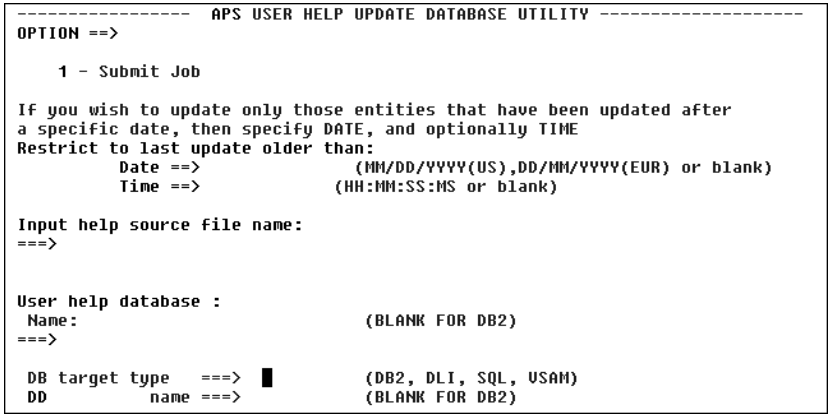
When you update the help database, you store the help source file you created.

Loading Help Source for VSAM

If your user application target is VSAM, perform the following steps:

1 Execute A1UHUPD to update the help database. Enter 2 in the Command field. From the Dictionary Services screen, enter 6 in the Command field. From the User Help Facility screen, enter 2 in the Command field. A1UHUPD displays the Update Database Utility screen where you provide parameters to execute this program.

Figure 8-8. APS User Help Update Database Utility Screen



2 Complete the fields on the Update Database Utility screen as follows:

- To store help text that has been changed during a specific time frame, enter values in the Date and Time fields. To store help text for all user application components, leave these fields blank.
- In the Input Help Source File Name field, specify the name of the help source file that you want to store in the help database.
- Enter the name of the help database in the User Help Database Name field.
- Enter vsam in the DB target type field.
- Enter the environment variable, helpvsm, in the Environment Name field.
- Type 1 in the Option field and press enter to update the help database.

The APS User Help Facility produces a report that identifies the updated help components, and stores the report file in the job queue.

Loading Help Source for IMS

If your user application target is IMS, perform the following steps:

1 Execute A1UHUPD with the PSB HELPPSBL. Enter 2 in the Command field. From the Dictionary Services screen, enter 6 in the Command field. From the User Help Facility screen, enter 2 in the Command field. A1UHUPD displays the Update Database Utility screen where you provide parameters to execute this program.

2 Complete the fields as follows:

- To store help text that has been changed during a specific time frame, enter values in the Date and Time fields. To store help text for all user application components, leave these fields blank.
- In the Input Help Source File Name field, specify the name of the help source file that you want to store in the help database.
- Enter the VSAM file name of the help database in the Name field.
- Enter dli in the DB target type field. If your DB target is SQL, enter sql for both SQL and DB2 targets.
- Enter the DDNAME, helpims, in the Environment Name field.
- Type 1 in the Option field and press enter to update the help database.

Special Considerations

- You cannot load a global field with more than one context. For example, if a date field is defined to the data element list with multiple formats, only one format will be loaded in your help database.
- The User Help Facility does not support the date and time parameters when you upload to the help database.

Customizing the User Help Application

You customize help applications by setting values for the variables in the User Help Facility control file, APHLPIN. This file resides in the APS CNTL member. Edit this file to specify or change:

- Program and screen names (if naming conflicts exists)
- Internal and external storage database targets
- Subschema access used by the help database
- Database name and attributes
- Database field names--COBOL or native
- Attribute restoration
- Screen data storage options
- Data field length
- Global screen message field name
- Field help indicator string
- Date format
- PF key designations
- COBOL help invocation conditions
- APS-generated User Help comment suppression

To customize user help, perform the following:

1. Copy and rename APSPRE.APSLIB.APSREL.CNTL(APHLPIN) to PROJECT.GROUP.CNTL(membername).
2. Edit PROJECT.GROUP.CNTL(membername) to overwrite any variables set in APHLPIN.
3. Add the following statement to the top of your program:

```
% INCLUDE USERCNTL(membername)
```

Note: If you or your administrator has modified the user help control file since initially generating your user application, you must also

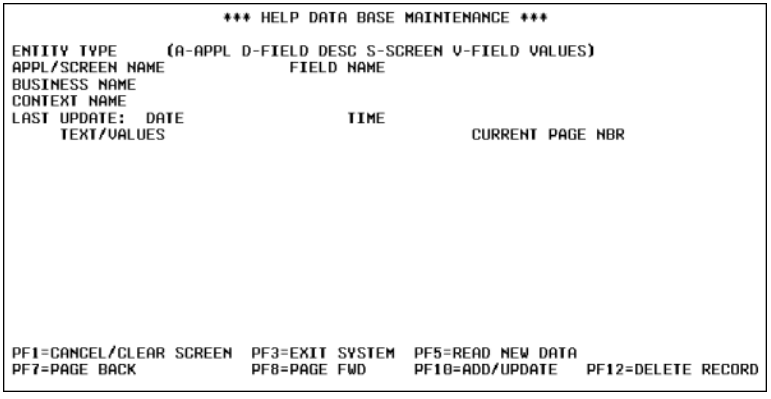
recompile the A1UHUPD, APSDISP, and APSFM programs of the APSUHELP application.

Maintaining the Help Database

APS provides an online file maintenance program, APSFM, that allows you to maintain the help database. When you execute APSFM, you can add, edit, or delete records stored in your help database. To do so, follow these steps:

1 In CICS , type *apsf*. In IMS, type /for apsfmo. In ISPF , you execute APSFM using the APS execution facilities. From the APS Main Menu, enter 3 in the Command field. APS displays the Prototype Execution screen. From this screen, enter 1 in the Command field. From the Prototype Execution screen, enter 1 in the Command field. Type apsfm in the program field. APS displays the Help Database Maintenance screen.

Figure 8-9. Help Database Maintenance Screen



Modify or delete help database records

2 To modify or delete a help database record, you must specify the help database record that you want to edit. To do so, type a value for the following fields as described below:

Field	Description	Value
Entity Type	Help database record to process	A = Application S = Screen D = Description V = Value of the data element
Entity Global Type	Specifies if the data element is global or local	global *applicationname* *screenname*
Name	Name of the entity	*dataelementname*

3 Press PF5. If the record exists, APS displays the existing data in the remaining fields. Edit the text using the ISPF line command keys listed on the screen.

4 If you are adding a new help database record, APS displays a message that the entity was not found. Complete the remaining fields as described below.

Field	Description
Business Name	Descriptive name of entity
Context Name	If data element is global enter a context name

5 Press PF10 to save the help database record, or PF3 to save the record and exit the program.

9 Define Online Programs with Program Painter

This chapter contains the following sections:

- *Concepts of the Program Painter*
- *Creating Online Programs in the Program Painter*

Concepts of the Program Painter

An alternative to Online Express

The Program Painter offers you a more conventional method for writing programs than Online Express. Unlike the menu-driven method of Online Express, the Program Painter method is text-driven, letting you enter your specifications on a blank screen using its ISPF-like text editor. Like Online Express, the Program Painter provides a shorthand method for creating online COBOL and COBOL/2 programs.

Mix online with batch programs

Your application can consist entirely of online programs, or you can mix online programs with batch programs in the same application. In addition, the programs of a single application can access different database and data communication (DB/DC) targets. For example, your online programs can use CICS to access SQL databases and VSAM files, while your batch programs access VSAM files and IMS databases. For all valid online and batch DB/DC target combinations, see *Paint the Application Definition*.

NTRY keyword generates a program template

By entering a single APS keyword, NTRY, you can generate a program template, or shell, that fully defines all parts of your program except for the procedural code that you supply. The template defines:

- The Identification Division, based on your Application Painter specifications
- The Environment Division, based on your Application Painter specifications

- The Data Division, including the following Working-Storage and Linkage Section structures:
  - Your database record or file definitions, based on your imported subschema
  - Your screen field data structures, based on your Screen Painter specifications
  - CICS EIBRCODE and DFHCOMMAREA structures
  - Your IMS PCB mask, including I/O and database PCBs, based on your imported subschema
  - An APS data structure for passing data among programs, known as a Commarea; the Commarea appears in either Working-Storage or the Linkage Section, depending on your DC target
  - PF key definitions, based on your specified DC target
  - Flags required by APS
- Portions of the Procedure Division, including:
  - A housekeeping routine, to initialize Working-Storage fields, flags, and counters
  - Program invocation logic, to initialize your program when it is invoked by a transaction ID, a screen, or another program, based on your specified DC target
  - Logic to send the program screen to the end user's monitor

Add to or modify the template

You can add to or modify the template as needed. When you do so, you enter additional APS keywords with your source code to specify the program location where the source belongs. For example, you can:

- Add Working-Storage or Linkage Section data elements and flags for your procedural routines.
- Redefine the APS Commarea data structure to accommodate the data that you pass among programs.
- Add to or modify the default program invocation logic to suit your program requirements.
- Add calls to user-defined Customization Facility macros, and set any variable values required by the macros.

Add procedural source code

To specify procedural logic, you can use any combination of the following types of source code:

- COBOL or COBOL/2
- APS database and data communications (DB and DC) calls. These calls provide almost all the functionality offered by your target environment calls, but are easier to write. The short, simple formats of the APS calls shield you from much tedious coding--you simply enter the call name and any keywords and arguments that you need. APS generates your specifications as complete calls, written in the syntax native to your DB and DC environments. For a complete list of calls for all DB and DC targets, see the"Database Calls" and "Database Communication Calls" topics in the APS Reference.
- APS Structured COBOL (S-COBOL) source code. S-COBOL is an optional set of COBOL-like procedural structures that are simpler and more powerful than COBOL or COBOL/2 structures. You can write S-COBOL statements in conjunction with, or instead of, COBOL or COBOL/2 statements. For information, see the "S-COBOL Structures" topic in the APS Reference.

Include external source code in your programs

In addition, you can include in your programs externally-defined source code that further streamlines the process of developing applications. When you do so, you enter additional APS keywords to specify the program location where the source code belongs. Applications created in the Program Painter can use any of the following types of external source code:

External Source Code	Data Set
Global stubs, which are COBOL, COBOL/2, or S-COBOL paragraphs that all programs of your application can share	APSPROG, your APS Project and Group data set for Program Painter programs and global stubs. You create stubs using the Program Painter; APS stores each stub in a separate file. For information on writing global stubs, see the "Stubs" topic in the APS Reference.
COBOL copybooks containing data structures or other source code	COPYLIB, your APS Project and Group data set for COBOL copybooks.

External Source Code	Data Set
Data structures created in the APS Data Structure Painter	APSDATA, your APS Project and Group data set for data structures that you create using the Data Structure Painter.
User-defined macros	USERMACS, your APS Project and Group data set for user-defined Customization Facility macros. For information on writing user-defined macros, see the APS Customization Facility User's Guide.

Columns for keywords and source code

You enter all program keywords and source code in the following columns of the Program Painter screen, depending on which compiler you use.

Compiler	Keyword Column Range	Source Code Column Range
OS/VS COBOL	8 through 11	12 through 72
COBOL/2	8 through 11	12 through 80

Creating Online Programs in the Program Painter

To create an online program using the Program Painter, follow these steps.

Create the application definition and screens

1 Create your application definition using the Application Painter, as described in *Paint the Application Definition*. Steps 2 and 3 below describe how to specify your DC and DB targets when creating your application definition.

2 Specify your DC target on the Application Painter as follows:

If application contains ...	Specify this DC target ...
Both online and batch programs	Your online DC target. To identify the batch programs, enter *batch in the Screen field next to each batch program name and leave the I/O fields blank.
Only online programs	Your online DC target.

3 Specify your database (DB) target in the DB field. For a list of valid DB/DC combinations for generating executable programs to run on various operating systems, see the "DB/DC Target Combinations" topic in the APS Reference.

To target DB/2, leave this field blank or let default to VSAM. Then, before generating the application, specify db2 in the SQL field on the Generation Options screen.

If your application accesses multiple database targets, specify a target as follows:

If application accesses ...	Specify this DB target ...
Two DB targets, including VSAM	The non-VSAM target, because APS always gives you access to the VSAM target.
Two or more DB targets, not including VSAM	Any of those DB targets. When you generate the programs, generate just the programs of your specified DB target first. Then change the DB target to the next target and generate just the programs of that next target. For example, if your application accesses both SQL and IMS subschemas, generate your SQL programs separately from your IMS programs.

4 Create your application screens using the Screen Painter, as described in *Paint Character Screens*.

Access the Program Painter

5 On the Application Painter, enter s next to a program name to display the Program Painter.

6 Begin entering your program source code. As you do so, specify the COBOL program locations where the code belongs--such as the Working-Storage Section or Procedure Division--by entering APS keywords next to the source code in the KYWD columns, 8 through 11. You can enter your source code and associated keywords in any sequence; when you generate the program, APS arranges the source into the proper COBOL program sequence. For example, you can define Working-Storage fields in the Procedure Division instead of Working-Storage.

Write Remarks

7 To write Identification Division Remarks text, enter the REM keyword in the KYWD column, and the text starting in column 12, on the same line. Continue on as many lines as you need. REM is invalid for COBOL/2; to write remarks in this environment, use the comment keyword instead, which is /*. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 REM     Comment text
         continues onto the next line.
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 /*      comment text
         continues onto the next line.
```

Specify Special-Names

8 To write an Environment Division Special-Names statement, enter the SPNM keyword in the KYWD column, and the statement starting in column 12, on the same line.

Define or include Working-Storage structures

9 Mark the beginning of your Working-Storage entries by entering the WS keyword in the KYWD column. Then skip a line and enter your Working-Storage structures--such as data structures, copylibs, and DB2 table and cursor declarations--as described in steps 10 through 15.

10 To define in Working-Storage a data structure in COBOL format, enter the 01 keyword in the KYWD column (columns 8 and 9), and your 01-level data item starting in column 12. To define elementary data items, skip a line and enter them starting in column 12, as shown below. We recommend that you indent each new level of elementary data items four columns. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 01      WS-STRUCT-IN-COBOL-FORMAT.
         05  MY-WS-FIELD-1                  PIC X(8).
         05  MY-WS-FIELD-2.
             10  MY-WS-FIELD-3              PIC X(8).
             10  MY-WS-FIELD-4              PIC X(3).
```

Generated APS source:

```
01  WS-STRUCT-IN-COBOL-FORMAT.
    05  MY-WS-FIELD-1                  PIC X(8).
    05  MY-WS-FIELD-2.
        10  MY-WS-FIELD-3              PIC X(8).
        10  MY-WS-FIELD-4              PIC X(3).
```

11 To define in Working-Storage a data structure in Data Structure Painter format, enter the REC keyword in the KYWD column (columns 8 through 10), and your 01-level data item starting in column 12. To define elementary data items, skip a line and enter them starting in column 16, as shown below. Do not enter the data item level numbers, such as 01 or 05; APS automatically generates them based on how you indent the items. We recommend that you indent each new level of elementary data items four columns. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 REC     WS-STRUCT-IN-DSPAINTER-FORMAT.
             MY-WS-FIELD-1                  X8
             MY-WS-FIELD-2
                 MY-WS-FIELD-3              X8
                 MY-WS-FIELD-3              X3
```

Generated APS source:

```
01  WS-STRUCT-IN-DSPAINTER-FORMAT.
    05  MY-WS-FIELD-1                  PIC X(8).
    05  MY-WS-FIELD-2.
        10  MY-WS-FIELD-3              PIC X(8).
        10  MY-WS-FIELD-4              PIC X(3).
```

12 To include a copybook in Working-Storage, choose one of the following methods:

- If you use a COBOL/2 compiler, or if your copybook contains an indexed table, enter the SYWS keyword in the KYWD column, and an APS % INCLUDE statement in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*--
 SYWS    % INCLUDE COPYLIB (MY-COPYBOOK)
```

- If you use an OS/VS COBOL compiler, and your copybook does not contain an indexed table, do one of the following:
  - Enter the 01 keyword in the KYWD column, and a COBOL COPY statement in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 01      COPY MY-COPYBOOK
```

  - Alternatively, copy the copybook into a Working-Storage field, as follows:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 01      WS-COPYBOOK-FLD COPY MY-COPYBOOK
```

13 To include in Working-Storage an externally-defined data structure defined in the Data Structure Painter, enter the DS keyword in the KYWD column, and the data structure name in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 DS      DATARECS
```

14 To define in Working-Storage a DB2 table declaration or one or more cursor declarations, enter the SQL keyword in the KYWD column, and the declaration(s) in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 SQL     DECLARE DSN8.TDEPT TABLE
         ... (DEPTNO CHAR(3) NOT NULL,
         ... DEPTNAME CHAR(36) NOT NULL,
         ... MGRNO CHAR(3) NOT NULL,
         ... ADMRDEPT CHAR(3) NOT NULL)
```

15 To include a PANVALET record in Working-Storage, enter the ++ keyword in the KYWD column, and the record name in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 ++      PANWSREC
```

Define or include Linkage Section structures

16 Mark the beginning of your Linkage Section entries by entering the LK keyword in the KYWD column. Skip a line and enter your

Linkage Section structures in the same manner that you enter Working-Storage structures.

Note: To include a copybook in the Linkage Section, substitute the SYWS keyword with the SYLK keyword.

Define the Commarea TP-USERAREA field

17 Accept or override the default length of the Commarea field, TP-USERAREA, and optionally redefine it into multiple fields. Although APS automatically generates a Commarea for all programs, its default length and program location vary by DC target, as shown below.

DC Target	Default TP-USERAREA Length	Program Location
CICS	80	Working-Storage
IMS	0	Working-Storage
ISPF	2048	Linkage Section
ISPF Dialog	0	Working-Storage

- To assign a length to TP-USERAREA, enter the variable value assignment statement, &TP-USER-LEN, in column 12, and the keyword SYM1 in the KYWD column. SYM1 places the variable at the top of your program. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*-
 SYM1    &TP-USER-LEN = 49
```

Generated APS source:

```
01  TP-COMMAREA.
    .
    .
    .
    05  TP-USERAREA            PIC X(49).
```

- To redefine TP-USERAREA, enter a redefinition data structure in either the Program Painter or Data Structure Painter, using either of the following keyword/source code combinations.

Keyword	Source Code
CA05	Define in the Program Painter a COBOL redefinition data structure. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*-
 SYM1    &TP-USER-LEN = 49
  .
  .
  .
 CA05    CA-REDEF.
         10  CA-EMPLOYEE-NAME     PIC X(20).
         10  CA-EMPLOYEE-TITLE    PIC X(20).
         10  CA-EMPLOYEE-SSN      PIC X(09).
```

Generated APS source:

```
 01  TP-COMMAREA.
     .
     .
     .
 01  FILLER REDEFINES TP-COMMAREA.
     05  TP-USERAREA                 PIC X(49).
     05  PGM-USERAREA    REDEFINES TP-USERAREA.
         10  CA-EMPLOYEE-NAME        PIC X(20).
         10  CA-EMPLOYEE-TITLE       PIC X(20).
         10  CA-EMPLOYEE-SSN         PIC X(09).
```

CA — Define in the Program Painter a redefinition data structure in Data Structure Painter format. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*-
 SYM1    &TP-USER-LEN = 49
  .
  .
  .
 CA      PGM-USERAREA
             10  CA-EMPLOYEE-NAME     X20
             10  CA-EMPLOYEE-TITLE    X20
             10  CA-EMPLOYEE-SSN      X09
```

The generated APS source is identical to the source generated by the CA05 keyword.

Keyword	Source Code
CADS	Define in the Data Structure Painter a redefinition data structure in Data Structure Painter format, and include it in the program using the CADS keyword. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*-
 CADS    PGM-USERAREA
```

Begin to define the Procedure Division

18 To begin defining the Procedure Division, enter the NTRY keyword in the KYWD column, and enter its arguments--such as the program screen--in column 12. NTRY generates logic to initialize your program when it is invoked, and to send the program's screen to the end-user's monitor, as shown below.

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 NTRY    PSINQY
```

Generated APS source:

```
003700 $TP-ENTRY ("PSINQY", "")
003710     IF TP-TRANSID-INVOKED
003720         PERFORM APS-TRANSID-INV-PARA
003730     ELSE-IF TP-PROGRAM-INVOKED
003740         PERFORM APS-PROGRAM-INV-PARA
003750     ELSE-IF TP-SCREEN-INVOKED
003760         PERFORM APS-SCREEN-INV-PARA
003770     $TP-SEND ( "PSINQY", "")
003780
003790 APS-TRANSID-INV-PARA.
003800     % IF &TP-USER-LEN > 0
003810         MOVE LOW-VALUES TO TP-USERAREA
003820     $SC-CLEAR ( "PSINQY")
003830     EJECT
003840 APS-PROGRAM-INV-PARA.
003850     $SC-CLEAR ( "PSINQY")
003860     EJECT
003870 APS-SCREEN-INV-PARA.
003880     PERFORM APS-USER-CODE-PARA
003890     EJECT
003900 APS-USER-CODE-PARA.
```

Enter Procedure Division source code

19 On the next line, enter your Procedure Division source code, which can include the following:

- COBOL, COBOL/2, or S-COBOL statements and paragraphs. To write any paragraph, enter the PARA keyword in the KYWD column, your paragraph name in column 12 on the same line, and your paragraph statements on the following lines. For

information on writing S-COBOL statements, see the "S-COBOL Structures" topic in the APS Reference.

- APS database and data communication (DB and DC) calls. For complete lists of calls for all DB and DC targets, see the "Database Calls" and "Data Communication Calls" topics in the APS Reference.
- COBOL, COBOL/2, or S-COBOL global stubs. To include a stub in the program, enter the STUB keyword in the KYWD column and your stub name in column 12 on the same line. For information on writing global stubs, see the "Stubs" topic in the APS Reference.
- Customization Facility macro calls and other statements. For information on writing these statements, see the Customization Facility User's Guide.

  For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*--
 NTRY    PSINQY
         /*  BEGIN PROCEDURE DIVISION SOURCE CODE
         IF PF12
             SEND PSMENU
         ELSE-IF PF1
             PSUPDT-EMPLOYEE-NO = PSINQY-EMPLOYEE-NO
             PSUPDT-FUNCTION = '1'
             SEND PSUPDT
         ELSE
             SEND PSINQY
         PERFORM SAMPLE-S-COBOL-PARA
         PERFORM SAMPLE-COBOL-PARA

 PARA    SAMPLE-COBOL-PARA.
         [I WILL INSERT SAMPLE COBOL PARA STMTS HERE]
 PARA    SAMPLE-S-COBOL-PARA
         [I WILL INSERT SAMPLE S-COBOL PARA STMTS HERE]
 STUB    MY-STUB
```

  Generated APS source:

```
003700 $TP-ENTRY ("PSINQY", "")
003710     IF TP-TRANSID-INVOKED
003720         PERFORM APS-TRANSID-INV-PARA
003730     ELSE-IF TP-PROGRAM-INVOKED
003740         PERFORM APS-PROGRAM-INV-PARA
003750     ELSE-IF TP-SCREEN-INVOKED
```

```
003760         PERFORM APS-SCREEN-INV-PARA
003770     $TP-SEND ( "PSINQY", "")
003780
003790 APS-TRANSID-INV-PARA.
003800     % IF &TP-USER-LEN > 0
003810         MOVE LOW-VALUES TO TP-USERAREA
003820     $SC-CLEAR ( "PSINQY")
003830     EJECT
003840 APS-PROGRAM-INV-PARA.
003850     $SC-CLEAR ( "PSINQY")
003860     EJECT
003870 APS-SCREEN-INV-PARA.
003880     PERFORM APS-USER-CODE-PARA
003890     EJECT
003900 APS-USER-CODE-PARA.
003910 /*  BEGIN PROCEDURE DIVISION SOURCE CODE
003920 IF PF12
003930     $TP-SEND PSMENU
003940 ELSE-IF PF1
003950     PSUPDT-EMPLOYEE-NO = PSINQY-EMPLOYEE-NO
003960     PSUPDT-FUNCTION = '1'
003970     $TP-SEND PSUPDT
003980 ELSE
003990     $TP-SEND PSINQY
004000 PERFORM SAMPLE-COBOL-PARA
004010 PERFORM SAMPLE-S-COBOL-PARA
004020 PERFORM MY-STUB
004030
004040 SAMPLE-COBOL-PARA.
004050     [COBOL PARA STMTS HERE]
004060 SAMPLE S-COBOL-PARA
004070     [COBOL PARA STMTS HERE]
004080 MY-STUB
004090     [SAMPLE STUB STMTS HERE]
```

Write comments **20** To document your program with comments, use the following formats in the following program locations. Note that in the

Procedure Division, you can enter comments at the end of a line of source code.

Program Location	Comment Format

Anywhere	`-KYWD-  12-*----20---*----30---*----40---*--`
	`/*  comment text`
	`/*  comment text`

Procedure Division	`-KYWD-  12-*----20---*----30---*----40---*--`
	`/* comment text`
	`program source code /* comment text`

Enter Customization Facility macro calls and statements

21 Enter any Customization Facility macro calls or statements that your program requires. For example, if on the Application Painter you include a user-defined macro library in your program, call the macros you need, and assign values to any variables that the macros require. Use the following keywords to place the calls and statements in the following program locations:

Keyword	Program Location
SYM1	At the beginning of the program, before macro libraries that you include at the beginning of the program
SYM2	After macro libraries that you include at the beginning of the program
SYEN	In the Environment Division, after the Special-Names paragraph
SYDD	At the beginning of the Data Division
SYFD	In the File Section, after macro libraries that you include at the beginning of the File Section
SYWS	In the Working-Storage Section, after macro libraries and data structures that you include in Working-Storage
SYLT	In the Linkage Section, after macro libraries and data structures that you include at the beginning of Linkage
SYLK	In the Linkage Section, after source code that you include with the SYLT keyword
SYBT	At the end of the program

For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 SYM1
 /*      MACRO VARIABLE TO APPEAR AT BEGINNING OF PROGRAM,
 /*       BEFORE MACRO LIBRARY THAT I INCLUDE AT BEGINNING
 /*       OF PROGRAM.
          % &MY-SYMBOL = 1234

 SYM2
 /*       MACRO VARIABLE TO APPEAR AFTER MACRO LIBRARY THAT
 /*       I INCLUDE AT BEGINNING OF PROGRAM.
          % &MY-STRING-SYMB = "THIS IS A STRING"

 SYWS
 /*       MACRO VARIABLE TO APPEAR AFTER MACRO LIBRARY THAT
 /*       I INCLUDE AT BEGINNING OF WORKING-STORAGE.
          % &MY-WS-SYMBOL = 1234
```

Validate source code syntax

22 To validate that your source code contains no Program Painter syntax errors, enter validate or val in the Command field. APS displays a message for each syntax error.

Preview the program as generated source

23 To preview the program as it will look when generated, enter convert or conv in the Command field. APS converts the Program Painter source code to generated APS source code. APS does not include in the converted source any components defined externally to the program; APS includes them when you generate the program. To view the source in Program Painter format again, enter reset or unconv. Such externally-defined components not included in this step are:

Component	Project\Group data set
Screen record descriptions	APSSCRN
Database record definitions	DDISYMB and COPYLIB
Data structures included from copylibs	COPYLIB
Data structures included from the Data Structure Painter	APSDATA
User-defined macros	USERMACS

To view the source in Program Painter format again, enter reset or unconv.

24 Exit the Program Painter by pressing PF3 or entering cancel.

Special Considerations

- When modifying your program, do not modify the generated source code; modify only your Program Painter source code.
- To customize the program template, you can write any custom source code and execute it at several predefined locations in the template. To do so, write paragraphs anywhere in the Procedure Division, using the APS-supplied paragraph names below. APS automatically performs the paragraphs at the locations specified below, in the following order:

Paragraph	Location Performed
APS-AFTER-RECEIVE-PARA	After the program is invoked by any method, and after all field edits are executed.
APS-TRANSID-INV-PARA	After the program is invoked by a transaction ID, and after all field edits are executed.
APS-PROGRAM-INV-PARA	After the program is invoked by another program.
APS-BEFORE-SEND-PARA	Before the program sends its screen, except when the program is invoked by another program.

10 Create Batch Programs

This chapter contains the following sections:

- *Concepts of APS Batch Programming*
- *Creating Batch Programs*
- *Sample Batch Program*

Concepts of APS Batch Programming

Shorthand method for creating programs

You create batch programs using the Program Painter, a tool that offers a shorthand method for developing applications. To fully define all divisions of your program except the Procedure Division, you simply enter APS keywords and their arguments. To help you create the Procedure Division more quickly, APS lets you write your database calls in simplified APS formats, saving you many lines of coding. You complete your Procedure Division by entering COBOL, COBOL/2, or S-COBOL structures. S-COBOL is an optional set of COBOL-like statements that simplify procedural coding. You enter all your program source code--including the APS keywords, database calls, and S-COBOL structures--on a blank Program Painter screen using its ISPF-like text editor.

Write reports using Report Writer structures

You can also use the Program Painter to create batch report programs using the APS Report Writer structures. This chapter only discusses creating batch programs that generate to a flat file. For information about writing reports, see *Create Reports with Report Writer*.

Mix batch with online programs

Your application can consist entirely of batch programs, or you can mix batch programs with online programs in the same application. In addition, the programs of a single application can access different database and data communication (DB/DC) targets. For example, your batch programs can access VSAM files and IMS databases, while your online programs use CICS to access VSAM files and SQL databases. For

all valid batch and online DB/DC target combinations, see *Paint the Application Definition*.

Keywords generate programs

APS builds batch program source code from the following items:

Identification Division:	Generated by APS, based on your Application Painter specifications
Environment Division:	Generated by APS keywords and arguments that you enter
Data Division File Section:	Generated by APS keywords and arguments that you enter
Data Division Working-Storage Section:	Generated by:
Database record definitions	APS, based on your subschema
IMS database PCB mask	APS, based on your subschema
Flags required by APS	APS
Data elements and flags for your procedural routines	APS keywords and source code that you enter
Data Division Linkage Section:	Generated by:
Data elements and flags for your procedural routines	APS keywords and source code that you enter
Procedure Division:	Generated by:
Routines to initialize APS Working-Storage flags	APS
Procedural source code	APS keywords, database calls, and other procedural source code that you enter

Add procedural source code

To specify procedural logic, you can use any combination of the following types of source code.

- COBOL or COBOL/2
- APS database (DB) calls. These calls provide almost all the functionality offered by your target environment calls, but are easier to write. The short, simple formats of the APS calls shield you from much tedious coding--you simply enter the call name and any keywords and arguments that you need. APS generates your specifications as complete calls, written in the syntax native to your

DB environment. For a complete list of calls for all DB targets, see the "About Database Calls" topic in the APS Reference.

- APS Structured COBOL (S-COBOL) source code. S-COBOL is an optional set of COBOL-like procedural structures that are simpler and more powerful than COBOL or COBOL/2 structures. You can write S-COBOL statements in conjunction with, or instead of, COBOL or COBOL/2 statements. For information, see the "About S-COBOL Structures" topic in the APS Reference.

Include external source code in your programs

In addition, you can include in your programs externally-defined source code that further streamlines the process of developing applications. When you do so, you enter additional APS keywords to specify the program location where the source code belongs. Applications created in the Program Painter can use any of the following types of external source code:

External Source Code	Data Set
Global stubs, which are COBOL, COBOL/2, or S-COBOL paragraphs that all programs of your application can share	APSPROG, your APS Project and Group data set for Program Painter programs and global stubs. You create stubs using the Program Painter; APS stores each stub in a separate file. For information on writing global stubs, see the "Stubs" topic in the APS Reference.
COBOL copybooks containing data structures or other source code	COPYLIB, your APS Project and Group data set for COBOL copybooks.
Data structures created in the APS Data Structure Painter	APSDATA, your APS Project and Group data set for data structures that you create using the Data Structure Painter.
User-defined macros	USERMACS, your APS Project and Group data set for user-defined Customization Facility macros. For information on writing user-defined macros, see the Customization Facility User's Guide.

Columns for keywords and source code

You enter all program keywords and source code in the following columns of the Program Painter screen, depending on which compiler you use.

Compiler	Keyword Column Range	Source Code Column Range
OS/VS COBOL	8 through 11	12 through 72
COBOL/2	8 through 11	12 through 80

Creating Batch Programs

To create a batch program using the Program Painter, follow these steps:

1 Create your application definition using the Application Painter, as described in *Paint the Application Definition*. Steps 2 and 3 below describe how to specify your DC and DB targets when creating your application definition.

2 Specify your DC target on the Application Painter as follows:

If application contains ...	Specify this DC target ...
Both batch and online programs	Your online DC target. To identify the batch programs, enter *batch in the Screen field next to each batch program name and leave the I/O fields blank.
Only batch programs	Mvs. Additionally, leave each Screen field and I/O field blank.

3 Specify your database (DB) target in the DB field. For a list of valid DB/DC combinations for generating executable programs to run on various operating systems, see the "DB/DC Target Combinations" topic in the APS Reference.

To target DB/2, leave this field blank or let default to VSAM. Then, before generating the application, specify db2 in the SQL field on the Generation Options screen.

If your application accesses multiple database targets, specify a target as follows:

If application accesses ...	Specify this DB target ...
Two DB targets, including VSAM	The non-VSAM target, because APS always gives you access to the VSAM target.
Two or more DB targets, not including VSAM	Any of those DB targets. When you generate the programs, first generate just the programs of your specified DB target. Then change the DB target to the next target and generate just the programs of that next target. For example, if your application accesses both VSAM and IMS subschemas, generate your VSAM programs separately from your IMS programs.

4 On the Application Painter, enter s next to a program name to display the Program Painter.

5 Begin entering your program source code. As you do so, specify the COBOL program locations where the code belongs--such as the Input-Output Section or File Section--by entering APS keywords next to the source code in the KYWD columns, 8 through 11. You can enter your source code and associated keywords in any sequence; when you generate the program, APS arranges the source into the proper COBOL program sequence. For example, you can define Working-Storage fields in the Procedure Division instead of Working-Storage.

Write Remarks

6 To write Identification Division Remarks text, enter the REM keyword in the KYWD column, and the text starting in column 12, on the same line. Continue on as many lines as you need. REM is invalid for COBOL/2; to write remarks in this environment, use the comment keyword /* instead. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 REM     Comment text
         continues onto the next line.
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 /*      Comment text
 /*      continues onto the next line.
```

Specify Special-Names

7 To write an Environment Division Special-Names statement, enter the SPNM keyword in the KYWD column, and the statement starting in column 12, on the same line.

Define File-Control

8 Define the Input-Output Section's File-Control paragraph as follows. For each input and output file, enter the IO keyword in the KYWD column, and, starting in column 12, enter the paragraph clauses. Do not enter the word SELECT in the SELECT clause; APS generates it for you. Continue on as many lines as you need. APS generates the Input-Output Section and File-Control headers. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 IO      INPUT-CUSTFILE
         ASSIGN TO EXTERNAL GARYDD
         ORGANIZATION IS LINE SEQUENTIAL
 IO      OUTPUT-FILE
         ASSIGN TO EXTERNAL GARYOUT
         ORGANIZATION IS LINE SEQUENTIAL
```

Generated APS source:

```
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT INPUT-CUSTFILE
     ASSIGN TO EXTERNAL GARYDD
     ORGANIZATION IS LINE SEQUENTIAL.
     SELECT OUTPUT-FILE
     ASSIGN TO EXTERNAL GARYOUT
     ORGANIZATION IS LINE SEQUENTIAL.
```

Define input file description

9 Define the file description of your first (or only) input file in the File Section as follows. Enter the FD keyword in the KYWD column, and a COBOL file description starting in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 FD      INPUT-CUSTFILE
         RECORD CONTAINS 80 CHARACTERS.
```

Define input file record description

10 To define the File Section input file's record description in COBOL format, enter the 01 keyword in the KYWD column (columns 8 and 9), and the 01-level data item starting in column 12. To define elementary data items, skip a line and enter them starting in column 12, as shown below. We recommend that you indent each new level of elementary data items four columns.

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 01      INPUT-REC.
         05  INP-ACTION-CODE     PIC X(1).
```

```
05  INP-CUSTOMER-NO     PIC X(6).
05  INP-CUSTOMER-NAME   PIC X(20).
05  INP-CUSTOMER-ADDR   PIC X(20).
05  INP-CUSTOMER-CITY   PIC X(20).
05  INP-CUSTOMER-ZIP    PIC X(9).
05  FILLER              PIC X(4).
```

11 To define the File Section input file's record description in Data Structure Painter format, enter the REC keyword in the KYWD column (columns 8 through 10), and the 01-level data item starting in column 12. To define elementary data items, skip a line and enter them starting in column 16, as shown below. Do not enter the data item level numbers, such as 01 or 05; APS automatically generates them based on how you indent the items. We recommend that you indent each new level of elementary data items four columns.

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 REC     INPUT-REC
             INP-ACTION-CODE     X1
             INP-CUSTOMER-NO     X6
             INP-CUSTOMER-NAME   X20
             INP-CUSTOMER-ADDR   X20
             INP-CUSTOMER-CITY   X20
             INP-CUSTOMER-ZIP    X9
             FILLER              X4
```

Generated APS source:

```
01  INPUT-REC.
    05  INP-ACTION-CODE     PIC X(1).
    05  INP-CUSTOMER-NO     PIC X(6).
    05  INP-CUSTOMER-NAME   PIC X(20).
    05  INP-CUSTOMER-ADDR   PIC X(20).
    05  INP-CUSTOMER-CITY   PIC X(20).
    05  INP-CUSTOMER-ZIP    PIC X(9).
    05  FILLER              PIC X(4).
```

12 If you created the File Section input file's record description using the Data Structure Painter, include the data structure in your program as follows. Enter the DS keyword in the KYWD column, and the data structure file name in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 DS      INREC
```

13 Define any additional input files in the File Section, in the same manner that you defined the first one.

Define output file description

14 Define the file description of your first (or only) output file in the File Section as follows. Enter the FD keyword in the KYWD column, and a COBOL file description starting in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 FD      OUTPUT-FILE
         RECORD CONTAINS 80 CHARACTERS.
```

Define output file record description

15 Define the File Section output file's record description in the same manner that you defined the input file's record description, as described in steps 10 through 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 01      OUTPUT-REC.
         05  OUTPUT-STATUS           PIC X(2).
         05  OUTPUT-CUSTOMER-NO      PIC X(6).
         05  OUTPUT-CUSTOMER-NAME    PIC X(20).
         05  OUTPUT-CUSTOMER-ADDR    PIC X(20).
         05  OUTPUT-CUSTOMER-CITY    PIC X(20).
         05  OUTPUT-CUSTOMER-ZIP     PIC X(9).
         05  OUTPUT-FILLER           PIC X(3).
```

16 Define any additional output files in the File Section, in the same manner that you defined the first one.

Define sort file description

17 To define a sort file description in the File Section, enter the SD keyword in the KYWD column, and the file description starting in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 SD      SORT-FILE
         RECORD CONTAINS 80 CHARACTERS
         DATA RECORD IS SORT-RECORD.
```

Define sort file record description

18 Define the File Section sort file's record description in the same manner that you defined those of the input and output files. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 01      SORT-RECORD.
         05  SORT-CUSTOMER-NAME      PIC X(8).
         05  FILLER                  PIC X(72).
```

Define or include Working-Storage structures

19 Mark the beginning of your Working-Storage entries by entering the WS keyword in the KYWD column. Then skip a line and enter your Working-Storage structures--such as data structures, copylibs, and DB2 table and cursor declarations--as described below in steps 20 through 25.

20 To define in Working-Storage a data structure in COBOL format, enter the 01 keyword in the KYWD column (columns 8 and 9), and your 01-level data item starting in column 12. To define elementary data items, skip a line and enter them starting in column 12, as shown below. We recommend that you indent each new level of elementary data items four columns. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 01      WS-STRUCT-IN-COBOL-FORMAT.
         05  MY-WS-FIELD-1                  PIC X(8).
         05  MY-WS-FIELD-2.
             10  MY-WS-FIELD-3              PIC X(8).
             10  MY-WS-FIELD-4              PIC X(3).
```

Generated APS source:

```
 01  WS-STRUCT-IN-COBOL-FORMAT.
     05  MY-WS-FIELD-1                  PIC X(8).
     05  MY-WS-FIELD-2.
         10  MY-WS-FIELD-3              PIC X(8).
         10  MY-WS-FIELD-4              PIC X(3).
```

21 To define in Working-Storage a data structure in Data Structure Painter format, enter the REC keyword in the KYWD column (columns 8 through 10), and the 01-level data item starting in column 12. To define elementary data items, skip a line and enter them starting in column 16, as shown below. Do not enter the data item level numbers, such as 01 or 05; APS automatically generates them based on how you indent the items. We recommend that you indent each new level of elementary data items four columns. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 REC     WS-STRUCT-IN-DSPAINTER-FORMAT
             MY-WS-FIELD-1                   X8
             MY-WS-FIELD-2
                 MY-WS-FIELD-3               X8
                 MY-WS-FIELD-3               X3
```

Generated APS source:

```
 01  WS-STRUCT-IN-DSPAINTER-FORMAT.
     05  MY-WS-FIELD-1                  PIC X(8).
     05  MY-WS-FIELD-2.
         10  MY-WS-FIELD-3              PIC X(8).
         10  MY-WS-FIELD-4              PIC X(3).
```

22 To include a copybook in Working-Storage, choose one of the following methods:

- If you use a COBOL/2 compiler, or if your copybook contains an indexed table, enter the SYWS keyword in the KYWD column, and an APS % INCLUDE statement in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*--
 SYWS    % INCLUDE COPYLIB (MY-COPYBOOK)
```

- If you use an OS/VS COBOL compiler, and your copybook does not contain an indexed table, do one of the following:

  - Enter the 01 keyword in the KYWD column, and a COBOL COPY statement in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*--
 01      COPY MY-COPYBOOK
```

  - Alternatively, copy the copybook into a Working-Storage field, as follows:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*--
 01      WS-COPYBOOK-FLD COPY MY-COPYBOOK
```

23 To include in Working-Storage an externally-defined data structure defined in the Data Structure Painter, choose one of the following methods:

- On the Application Painter, enter the data structure name in the Data Str(ucture) field, and ws in the Loc(ation) field.

- Enter the DS keyword in the KYWD column, and the data structure file name in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*--
 DS      DATARECS
```

24 To define in Working-Storage a DB2 table declaration or one or more cursor declarations, enter the SQL keyword in the KYWD column, and the declaration(s) in column 12. For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 SQL     DECLARE DSN8.TDEPT TABLE
         ... (DEPTNO CHAR(3) NOT NULL,
         ... DEPTNAME CHAR(36) NOT NULL,
         ... MGRNO CHAR(3) NOT NULL,
         ... ADMRDEPT CHAR(3) NOT NULL)
```

25 To include a PANVALET record in Working-Storage, enter the ++ keyword in the KYWD column, and the record name in column 12. For example:

```
-KYWD-    12-*----20---*----30---*----40---*----50---*----
 ++       PANWSREC
```

Define or include Linkage Section structures

26 If your program receives data from a calling program, define its Linkage Section as follows. Mark the beginning of your Linkage Section entries by entering the LK keyword in the KYWD column. Skip a line and enter your Linkage Section structures in the same manner that you enter Working-Storage structures.

Note: To include a copybook in the Linkage Section, substitute the SYWS keyword, as shown in step 22, with the SYLT or SYLK keyword.

Begin to define the Procedure Division

27 Mark the beginning of the Procedure Division by entering either the PROC or NTRY keyword to generate the PROCEDURE DIVISION statement appropriate for your program, as shown below.

Program Type	Procedure Division Keyword
Calling or non-calling	NTRY or PROC; both keywords generate a PROCEDURE DIVISION statement.
Called	PROC with optional USING clause; generates a PROCEDURE DIVISION USING statement, enabling the program to receive data items from a calling program' s CALL statement.
Any IMS program	NTRY. PROC is invalid. To specify arguments for a PROCEDURE DIVISION USING clause, specify them in a TP-LINKAGE call that you code in the Linkage Section.

For example, enter the PROC keyword with the USING clause data items TOTAL, W-BALANCE, and CHARGERECORD to generate a PROCEDURE DIVISION USING statement in a called program, as shown below.

```
-KYWD-    12-*----20---*----30---*----40---*----50---*----
 PROC     TOTAL W-BALANCE CHARGERECORD
```

Generated APS source:

```
PROCEDURE DIVISION USING TOTAL W-BALANCE CHARGERECORD.
```

Enter Procedure Division source code

28 On the next line, enter your Procedure Division source code, which can include the following:

- COBOL, COBOL/2, or S-COBOL statements and paragraphs. To write any paragraph, enter the PARA keyword in the KYWD column, your paragraph name in column 12 on the same line, and your paragraph statements on the following lines. For information on writing S-COBOL statements, see the "S-COBOL Structures" topic in the APS Reference.
- APS database (DB) calls. For a complete list of calls for all DB targets, see the "Database Calls" topic in the APS Reference.
- COBOL, COBOL/2, or S-COBOL global stubs. To include a stub in the program, enter the STUB keyword in the KYWD column and your stub name in column 12 on the same line. For information on writing global stubs, see the "Stubs" topic in the APS Reference.
- Customization Facility macro calls and other statements. For information on writing these statements, see the APS Customization Facility User's Guide.

  For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*--
 PROC
         OPEN INPUT INPUT-CUSTFILE
         ...  OUTPUT OUTPUT-FILE
         REPEAT
             READ INPUT-CUSTFILE
             WS-CUST-NO = INP-CUSTOMER-NO
         UNTIL AT END ON INPUT-CUSTFILE
             EVALUATE INP-ACTION-CODE
             WHEN 'Q'
                 PERFORM QUERY-LOGIC
             WHEN 'U'
                 PERFORM UPDATE-LOGIC
             WHEN 'D'
                 PERFORM DELETE-LOGIC
         CLOSE INPUT-CUSTFILE
         ...   OUTPUT-FILE

 PARA    QUERY-LOGIC
```

```
DB-OBTAIN REC CUSTOMER-REC
... WHERE CM_CUSTOMER_NO = #WS-CUST-NO
IF OK-ON-REC
    OUTPUT-STATUS      = 'SQ'
    PERFORM MOVE-COPYLIB-TO-OUTPUT
    PERFORM WRITE-MSGOUT
ELSE
    OUTPUT-STATUS      = 'UQ'
    PERFORM MOVE-INPUT-TO-OUTPUT
    PERFORM WRITE-MSGOUT
.
.
```

Define Declaratives Section

29 To define a Declaratives Section, choose one of the following methods:

- To specify Declaratives Section sections and paragraphs, use the DPAR keyword, as shown below. Do not enter the DECLARATIVES header; APS generates it. APS also generates the END DECLARATIVES statement at the appearance of another keyword in the KYWD column; be sure that a keyword appears at the end of your Declaratives Section.

```
-KYWD-   12-*----20---*----30---*----40---*----50---*--
 DPAR    section-1-name SECTION declarative-sentence
 DPAR    para-1-name
         /* para-1-name source code
         .
         .
         .
 DPAR    section-2-name SECTION declarative-sentence
 DPAR    para-2-name
         /* para-2-name source code
         .
         .
```

Generated APS source:

```
DECLARATIVES.

section-1-name SECTION. declarative-sentence
para-1-name.
/* para-1-name source code
.
.
.
section-2-name SECTION. declarative-sentence
para-2-name.
```

```
/* para-2-name source code
.
.

END DECLARATIVES.
```

- To specify Declaratives Section statements only--not sections or paragraphs--use the DECL keyword, as shown below.

```
-KYWD-   12-*----20---*----30---*----40---*----50---*--
 DECL    declarative-statement
         declarative-statement
```

Generated APS source:

```
DECLARATIVES.

declarative-statement
declarative-statement

END DECLARATIVES.
```

Write comments **30** To document your program with comments, use the following formats in the following program locations. Note that in the Procedure Division, you can enter comments at the end of a line of source code.

Program Location	Comment Format

Anywhere	`-KYWD-  12-*----20---*----30---*----40---*--`
	` /*  comment text`
	` /*  comment text`

Procedure Division	`-KYWD-  12-*----20---*----30---*----40---*--`
	`/* comment text`
	`program source code /* comment text`

Enter Customization Facility macro calls and statements **31** Enter any Customization Facility macro calls or statements that your program requires. For example, if on the Application Painter you include a user-defined macro library in your program, you should call the macros you need, and assign values to any variables that the

macros require. Use the following keywords to place the calls and statements in the following program locations:

Keyword	Program Location
SYM1	At the beginning of the program, before macro libraries that you include at the beginning of the program
SYM2	After macro libraries that you include at the beginning of the program
SYEN	In the Environment Division, after the Special-Names paragraph
SYIO	In the Input-Output Section, after macro libraries that you include at the beginning of the Input-Output Section
SYDD	At the beginning of the Data Division
SYFD	In the File Section, after macro libraries that you include at the beginning of the File Section
SYWS	In the Working-Storage Section, after macro libraries and data structures that you include in Working-Storage
SYLT	In the Linkage Section, after macro libraries and data structures that you include at the beginning of Linkage
SYLK	In the Linkage Section, after source code that you include with the SYLT keyword
SYRP	In the Report Section, after any macro libraries that you include at the beginning of the Report Section
SYBT	At the end of the program

For example:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 SYM1
 /*      MACRO VARIABLE TO APPEAR AT BEGINNING OF PROGRAM,
 /*      AFTER MACRO LIBRARY THAT I INCLUDE AT BEGINNING
 /*      OF PROGRAM.
         % &REC-LEN = 80
 SYFD
 /*      MACRO CALL TO APPEAR IN FILE SECTION, AFTER MACRO
 /*      LIBRARY THAT I INCLUDE AT BEGINNING OF FILE
 /*      SECTION.
         % $INPUTFILE-REC-DESCRIP( 'INPUT-REC')
```

Validate source code syntax

32 To validate that your source code contains no Program Painter syntax errors, enter validate or val in the Command field. APS displays a message for each syntax error.

Preview the program as generated source

33 To preview the program as it will look when generated, enter convert or conv in the Command field. APS converts the Program Painter source code to generated APS source code. APS does not include in the converted source any components defined externally to the program; APS includes them when you generate the program. Such externally-defined components not included at this step are:

Component	Project\Group Data Set
Database record definitions	DDISYMB and COPYLIB
Data structures included from copylibs	COPYLIB
Data structures included from the Data Structure Painter	APSDATA
User-defined macros	USERMACS

To view the source in Program Painter format again, enter reset or unconv.

34 Exit the Program Painter by pressing PF3 or entering cancel.

Special Consideration

When modifying your program, do not modify the generated source code; modify only your Program Painter source code.

Sample Batch Program

Below is a complete program illustrating many APS batch programming features.

Program Painter source:

```
-KYWD-   12-*----20---*----30---*----40---*----50---*----
 IO      INPUT-CUSTFILE
         ASSIGN TO GARYDD
```

```
        ORGANIZATION IS LINE SEQUENTIAL
IO      OUTPUT-FILE
        ASSIGN TO GARYOUT
        ORGANIZATION IS LINE SEQUENTIAL
FD      INPUT-CUSTFILE
        RECORD CONTAINS 80 CHARACTERS.
01      INPUT-REC.
        05  INP-ACTION-CODE     PIC X(1).
        05  INP-CUSTOMER-NO     PIC X(6).
        05  INP-CUSTOMER-NAME   PIC X(20).
        05  INP-CUSTOMER-ADDR   PIC X(20).
        05  INP-CUSTOMER-CITY   PIC X(20).
        05  INP-CUSTOMER-ZIP    PIC X(9).
        05  FILLER              PIC X(4).
FD      OUTPUT-CUSTFILE
        RECORD CONTAINS 80 CHARACTERS.
01      OUTPUT-REC.
        05  OUTPUT-STATUS           PIC X(2).
        05  OUTPUT-CUSTOMER-NO      PIC X(6).
        05  OUTPUT-CUSTOMER-NAME    PIC X(20).
        05  OUTPUT-CUSTOMER-ADDR    PIC X(20).
        05  OUTPUT-CUSTOMER-CITY    PIC X(20).
        05  OUTPUT-CUSTOMER-ZIP     PIC X(9).
        05  OUTPUT-FILLER           PIC X(3).

PROC
        OPEN INPUT INPUT-CUSTFILE
        ...  OUTPUT OUTPUT-FILE
        REPEAT
            READ INPUT-CUSTFILE
            WS-CUST-NO = INP-CUSTOMER-NO
        UNTIL AT END ON INPUT-CUSTFILE
            EVALUATE INP-ACTION-CODE
            WHEN 'Q'
                PERFORM QUERY-LOGIC
            WHEN 'U'
                PERFORM UPDATE-LOGIC
            WHEN 'D'
                PERFORM DELETE-LOGIC
        CLOSE INPUT-CUSTFILE
        ...   OUTPUT-FILE

PARA    QUERY-LOGIC
        DB-OBTAIN REC CUSTOMER-REC
        ... WHERE CM_CUSTOMER_NO = #WS-CUST-NO
        IF OK-ON-REC
            OUTPUT-STATUS      = 'SQ'
            PERFORM MOVE-COPYLIB-TO-OUTPUT
```

```
            PERFORM WRITE-MSGOUT
        ELSE
            OUTPUT-STATUS      = 'UQ'
            PERFORM MOVE-INPUT-TO-OUTPUT
            PERFORM WRITE-MSGOUT

PARA    UPDATE-LOGIC
        PERFORM MOVE-INPUT-TO-COPYLIB
        DB-MODIFY REC CUSTOMER-REC
        ... WHERE CM_CUSTOMER_NO = #WS-CUST-NO
        IF OK-ON-REC
            OUTPUT-STATUS      = 'SM'
            PERFORM MOVE-COPYLIB-TO-OUTPUT
            PERFORM WRITE-MSGOUT
        ELSE
            OUTPUT-STATUS      = 'UM'
            PERFORM MOVE-COPYLIB-TO-OUTPUT
            PERFORM WRITE-MSGOUT

PARA    DELETE-LOGIC
        DB-ERASE REC CUSTOMER-REC
        ... WHERE CM_CUSTOMER_NO = #WS-CUST-NO
        IF OK-ON-REC
            OUTPUT-STATUS      = 'SE'
            PERFORM MOVE-INPUT-TO-OUTPUT
            PERFORM WRITE-MSGOUT
        ELSE
            OUTPUT-STATUS      = 'UE'
            PERFORM MOVE-INPUT-TO-OUTPUT
            PERFORM WRITE-MSGOUT

PARA    ADD-LOGIC
        PERFORM MOVE-INPUT-TO-COPYLIB
        DB-STORE REC CUSTOMER-REC
        ... WHERE CM_CUSTOMER_NO = #WS-CUST-NO
        IF OK-ON-REC
            OUTPUT-STATUS      = 'SS'
            PERFORM MOVE-INPUT-TO-OUTPUT
            PERFORM WRITE-MSGOUT
        ELSE
            OUTPUT-STATUS      = 'BS'
            PERFORM MOVE-INPUT-TO-OUTPUT
            PERFORM WRITE-MSGOUT

PARA    MOVE-INPUT-TO-OUTPUT
        OUTPUT-STATUS                 = OUTPUT-STATUS
        OUTPUT-CUSTOMER-NO            = INP-CUSTOMER-NO
        OUTPUT-CUSTOMER-NAME          = INP-CUSTOMER-NAME
```

```
        OUTPUT-CUSTOMER-ADDR          = INP-CUSTOMER-ADDR
        OUTPUT-CUSTOMER-CITY          = INP-CUSTOMER-CITY
        OUTPUT-CUSTOMER-ZIP           = INP-CUSTOMER-ZIP

PARA    MOVE-COPYLIB-TO-OUTPUT
        OUTPUT-CUSTOMER-NO            = CM-CUSTOMER-NO
        OUTPUT-CUSTOMER-NAME          = CM-CUSTOMER-NAME
        OUTPUT-CUSTOMER-ADDR          = CM-CUSTOMER-ADDR
        OUTPUT-CUSTOMER-CITY          = CM-CUSTOMER-CITY
        OUTPUT-CUSTOMER-ZIP           = CM-CUSTOMER-ZIP

PARA    MOVE-INPUT-TO-COPYLIB
        CM-CUSTOMER-NO                = INP-CUSTOMER-NO
        CM-CUSTOMER-NAME              = INP-CUSTOMER-NAME
        CM-CUSTOMER-ADDR              = INP-CUSTOMER-ADDR
        CM-CUSTOMER-CITY              = INP-CUSTOMER-CITY
        CM-CUSTOMER-ZIP               = INP-CUSTOMER-ZIP

PARA    WRITE-MSGOUT
        WRITE OUTPUT-REC
WS
01      THEFLDS.
        05  WS-CUST-NO             PIC X(6).
```

Generated APS source:

```
%   &AP-GEN-VER = 2200
%   &AP-PGM-ID = "SAMPLPGM"
%   &AP-MAIN-PROGRAM- = "NO"
%   &AP-GEN-DC-TARGET = "MVS"
%   &AP-GEN-DB-TARGET = "VSAM"
%   &AP-GEN-USER-HELP = "NO"
%   &AP-PROC-DIV-KYWD-SEEN = 1
%   &AP-FILE-CONTROL-SEEN = 1
%   &AP-SUBSCHEMA = "SAMPLSUB"
%   &AP-APPLICATION-ID = "JOHND"
%   &AP-GEN-DATE = "930407"
%   &AP-GEN-TIME = "07244461"


%*  --- SUBSCHEMA / PSB FROM APPLICATION DEFINITION ---
$DB-SUBSCHEMA("SAMPLSUB")

IDENTIFICATION DIVISION.
PROGRAM-ID.                         SAMPLPGM.
AUTHOR.                             JOHND.
DATE-WRITTEN.                       93/04/07.
DATE-COMPILED.                      &COMPILETIME.
```

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER.                  &SYSTEM.
OBJECT-COMPUTER.                  &SYSTEM.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-CUSTFILE
    ASSIGN GARYDD
    ORGANIZATION IS LINE SEQUENTIAL.
    SELECT OUTPUT-FILE
    ASSIGN GARYOUT
    ORGANIZATION IS LINE SEQUENTIAL.

DATA DIVISION.

FILE SECTION.

FD  INPUT-CUSTFILE
    RECORD CONTAINS 80 CHARACTERS.
01  INPUT-REC.
    05  INP-ACTION-CODE     PIC X(1).
    05  INP-CUSTOMER-NO     PIC X(6).
    05  INP-CUSTOMER-NAME   PIC X(20).
    05  INP-CUSTOMER-ADDR   PIC X(20).
    05  INP-CUSTOMER-CITY   PIC X(20).
    05  INP-CUSTOMER-ZIP    PIC X(9).
    05  FILLER              PIC X(4).
FD  OUTPUT-FILE
    RECORD CONTAINS 80 CHARACTERS.
01  OUTPUT-REC
    05  OUTPUT-STATUS           PIC X(2).
    05  OUTPUT-CUSTOMER-NO      PIC X(6).
    05  OUTPUT-CUSTOMER-NAME    PIC X(20).
    05  OUTPUT-CUSTOMER-ADDR    PIC X(20).
    05  OUTPUT-CUSTOMER-CITY    PIC X(20).
    05  OUTPUT-CUSTOMER-ZIP     PIC X(9).
    05  OUTPUT-FILLER           PIC X(3).


WORKING-STORAGE SECTION.
$TP-WS-MARKER

01  THEFLDS.
    05  WS-CUST-NO              PIC X(6).
```

```
01  TEXT-MSG                    PIC X(30)
            VALUE &SQ+PLEASE ENTER NEXT TRANSID&SQ.

PROCEDURE DIVISION.
   OPEN INPUT INPUT-CUSTFILE
    ...  OUTPUT OUTPUT-FILE
   REPEAT
        READ INPUT-CUSTFILE
        WS-CUST-NO = INP-CUSTOMER-NO
    UNTIL AT END ON INPUT-CUSTFILE
        EVALUATE INP-ACTION-CODE
        WHEN 'Q'
            PERFORM QUERY-LOGIC
        WHEN 'U'
            PERFORM UPDATE-LOGIC
        WHEN 'D'
            PERFORM DELETE-LOGIC
    CLOSE INPUT-CUSTFILE
    ...   OUTPUT-FILE

QUERY-LOGIC
    $DB-OBTAIN ( "REC CUSTOMER-REC WHERE CM_CUSTOMER_NO = ",
    %... "#WS-CUST-NO" )
    IF OK-ON-REC
        OUTPUT-STATUS      = 'SQ'
        PERFORM MOVE-COPYLIB-TO-OUTPUT
        PERFORM WRITE-MSGOUT
    ELSE
        OUTPUT-STATUS      = 'UQ'
        PERFORM MOVE-INPUT-TO-OUTPUT
        PERFORM WRITE-MSGOUT

UPDATE-LOGIC
    PERFORM MOVE-INPUT-TO-COPYLIB
    $DB-MODIFY ( "REC CUSTOMER-REC WHERE CM_CUSTOMER_NO = ",
    %... "#WS-CUST-NO" )
    IF OK-ON-REC
        OUTPUT-STATUS      = 'SM'
        PERFORM MOVE-COPYLIB-TO-OUTPUT
        PERFORM WRITE-MSGOUT
    ELSE
        OUTPUT-STATUS      = 'UM'
        PERFORM MOVE-COPYLIB-TO-OUTPUT
        PERFORM WRITE-MSGOUT

DELETE-LOGIC
    $DB-ERASE ( "REC CUSTOMER-REC WHERE CM_CUSTOMER_NO = ",
    %... "#WS-CUST-NO" )
```

```
    IF OK-ON-REC
        OUTPUT-STATUS      = 'SE'
        PERFORM MOVE-INPUT-TO-OUTPUT
        PERFORM WRITE-MSGOUT
    ELSE
        OUTPUT-STATUS      = 'UE'
        PERFORM MOVE-INPUT-TO-OUTPUT
        PERFORM WRITE-MSGOUT

ADD-LOGIC
    PERFORM MOVE-INPUT-TO-COPYLIB
    $DB-STORE ( "REC CUSTOMER-REC WHERE CM_CUSTOMER_NO = ",
    %... "#WS-CUST-NO" )
    IF OK-ON-REC
        OUTPUT-STATUS      = 'SS'
        PERFORM MOVE-INPUT-TO-OUTPUT
        PERFORM WRITE-MSGOUT
    ELSE
        OUTPUT-STATUS      = 'BS'
        PERFORM MOVE-INPUT-TO-OUTPUT
        PERFORM WRITE-MSGOUT

MOVE-INPUT-TO-OUTPUT
    OUTPUT-STATUS                  = OUTPUT-STATUS
    OUTPUT-CUSTOMER-NO             = INP-CUSTOMER-NO
    OUTPUT-CUSTOMER-NAME           = INP-CUSTOMER-NAME
    OUTPUT-CUSTOMER-ADDR           = INP-CUSTOMER-ADDR
    OUTPUT-CUSTOMER-CITY           = INP-CUSTOMER-CITY
    OUTPUT-CUSTOMER-ZIP            = INP-CUSTOMER-ZIP

MOVE-COPYLIB-TO-OUTPUT
    OUTPUT-CUSTOMER-NO             = CM-CUSTOMER-NO
    OUTPUT-CUSTOMER-NAME           = CM-CUSTOMER-NAME
    OUTPUT-CUSTOMER-ADDR           = CM-CUSTOMER-ADDR
    OUTPUT-CUSTOMER-CITY           = CM-CUSTOMER-CITY
    OUTPUT-CUSTOMER-ZIP            = CM-CUSTOMER-ZIP

MOVE-INPUT-TO-COPYLIB
    CM-CUSTOMER-NO                 = INP-CUSTOMER-NO
    CM-CUSTOMER-NAME               = INP-CUSTOMER-NAME
    CM-CUSTOMER-ADDR               = INP-CUSTOMER-ADDR
    CM-CUSTOMER-CITY               = INP-CUSTOMER-CITY
    CM-CUSTOMER-ZIP                = INP-CUSTOMER-ZIP

WRITE-MSGOUT
    WRITE OUTPUT-REC
```

11 Create Reports with Report Writer

This chapter contains the following sections:

- *Concepts of APS Report Writing*
- *Painting Report Mock-Ups*
- *Creating Report Programs*
- *Generate Multiple SUM or SOURCE Statements*
- *Mapping Considerations*
- *Sample Program*

Concepts of APS Report Writing

Specify the physical report appearance

APS report writing features let you produce a report by specifying the physical appearance of the report, rather than the detailed procedures necessary to produce that report. Instead of writing COBOL statements that determine the relationship of output lines, recognize page overflow, construct headers and footers, recognize logical data groups, format output lines, map data to output fields, and perform data calculations, APS sets up the routines needed to produce the report in the requested format -- all you do is paint a visual representation of the report and specify which items control the report logic.

Paint report mock-ups

First, you paint the report layouts, called mock-ups, in the Report Painter, which provides a free-form definition facility. You can define the mock-up by typing literals and output fields to visually represent the report output. You can specify both floating numeric and alphanumeric output edit masks directly within the report mock-up.

Reuse report mock-ups

APS for z/OS automatically stores mock-ups in the Application Dictionary. The mock-ups are available as report templates or for use in multiple programs.

Define report logic

After you create a mock-up, you define the report logic in the Program Painter using APS Report Writer structures. Report Writer structures let you automatically perform paging, calculate field values, test and execute control breaks, generate multiple reports, and generate all logic necessary to map fields between reports and databases or files. You can include multiple reports in a single program.

Name the input and output files

Using the Program Painter IO and FD keywords, you name the input data file, the output report file, and the report itself.

Add a Report Section

With the Program Painter RED, MOCK, and 01 keywords, you add a Report Section naming the report and defining the format of each report named. There are two types of format entries:

- Those that describe the physical aspects of the report format, such as the maximum number of lines per page, where report lines appear on the page, and which data items are controls.
- Those that describe the function, format, and characteristics of each report line.

Determine your report groups

APS categorizes the report lines into report groups, which are groups of report lines that make up the headings, body, and footings of the report. Report groups include the following:

- Report Heading - Header lines that print once at the beginning of a report. Optionally, it can appear on a page by itself.
- Page Heading - Header lines that print at the top of each page.
- Control Heading - Header lines that print each time a control break occurs.
- Detail - Detail lines that are the body of the report. Detail lines are not required for summary reports.
- Control Footing - Line(s) of totals that print at the end of each detail group, immediately following the detail lines.
- Page Footing - Footer lines that print at the bottom of each page.
- Report Footing - Footer lines that print once at the end of a report.

Types of Report Groups shows a sample mock-up with the various report group types.

Types of Report Groups:

Report Heading	`WONDERFUL WIDGETS INCORPORATED STOCK REPORT XXXXXX XXX`
Page Heading	`MID-ATLANTIC STOCK REPORT XXXXXX XXX`
Control Heading	`LOCATION LAST COUNT QUANTITY QUANTITY QUANTITY DATE IN STOCK ISSUED RECEIVED`
Detail Line	`XXXXXXXXXXXX 99/99/99 ZZZ,ZZ9 ZZZ,ZZ9 ZZZ,ZZ9 --------- --------- ---------`
Control Footing	`TOTAL BY LOCATION: Z,ZZZ,ZZ9 Z,ZZZ,ZZ9 Z,ZZZ,ZZ9`

Control Footing Final	`TOTAL NUMBER OF SALES BY LOCATION: ZZZ,ZZ9`
	`TOTAL WONDERFUL WIDGETS IN STOCK: Z,ZZZ,ZZ9`
	`TOTAL WONDERFUL WIDGETS ISSUED: Z,ZZZ,ZZ9`
	`TOTAL WONDERFUL WIDGETS RECEIVED: Z,ZZZ,ZZ9`
	`TOTAL WONDERFUL WIDGETS SOLD: Z,ZZZ,ZZ9`

Page Footing	`PAGE ZZZ9`
Report Footing	`***** END OF REPORT *****`

Further defining detail lines

Entries for the detail lines which make up the body of the report describe the characteristics of the data items, such as the format, its placement in relation to the other data items, and any control factors. You can use the following statements to define the line contents:

- The SOURCE statement maps a data item to the report output field, using the current value of this data item each time the field prints.
- The REFERENCE statement identifies a non-printing data item for summing in a control footing.
- The SUM statement totals the values in the named fields. When a SUM statement executes, APS automatically:
  - Creates a Working-Storage SUM accumulator field for each data item.
  - Increments the SUM accumulator.
  - Prints the accumulated values at control break time.
  - Resets the SUM accumulator to zero after printing.

- The VALUE statement designates a literal value to print for the field each time the line prints.

Set control breaks

You use controls to specify how to arrange the data your report displays. For example, you might want to display detail lines arranged by sales territories within cities or states or both. You can have up to 28 control breaks. APS processes controls from the most inclusive down to the least inclusive, that is from major to minor. When a data item designated as a control changes value, such as a monthly change in a date field, a control break occurs, and APS does the following for you:

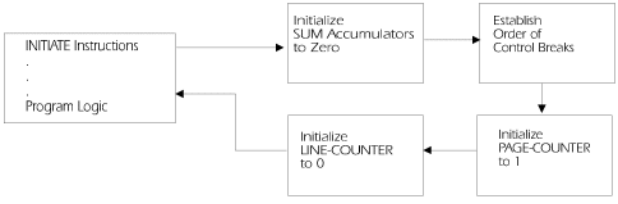
- Prints the detail line that caused the break.
- Prints control headings for any lower-level data items, followed by the heading for the data item that caused the control break.
- Prints control totals for the current and lower-level data items.
- Clears all associated counters and accumulators.

Produce the report in the Procedure Division

In the Procedure Division, you open your input and output files, execute and print the report, and close the files, using the following three Report Writer statements:

- The INITIATE statement performs functions in the Report Writer analogous to the OPEN statement for individual files. *INITIATE Logic Processing* illustrates INITIATE processing.

Figure 11-1. INITIATE Logic Processing



- The GENERATE statement produces the body of the report, and executes and prints the entire report. APS automatically does the following for you:
  - Prints specified headings and footings
  - Increments and resets counters and accumulators as necessary

- Obtains source information
- Produces sum information
- Moves values to the data item(s) in the report group entries
- Tests controls
- Prints detail lines
- Pages the report
- Prints the all lines required when a control break occurs

GENERATE Logic Processing and *GENERATE Logic Processing* illustrate GENERATE processing.

Figure 11-2. GENERATE Logic Processing

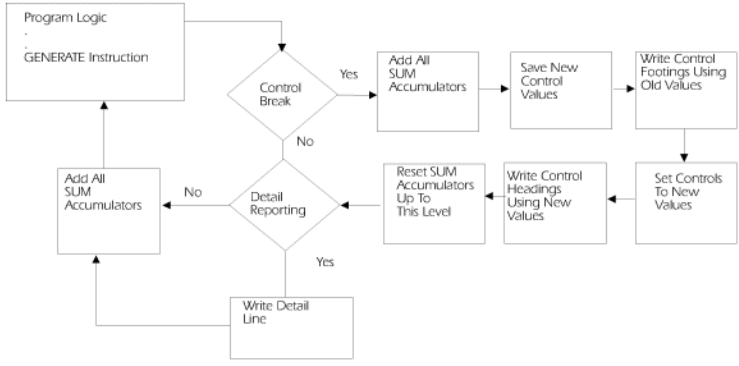
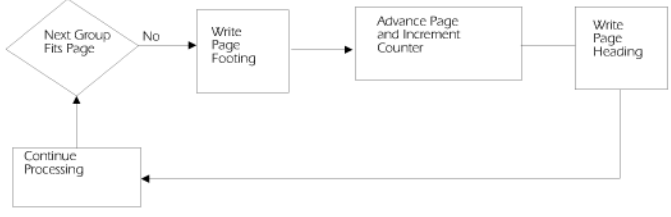


Figure 11-3. GENERATE Logic Processing

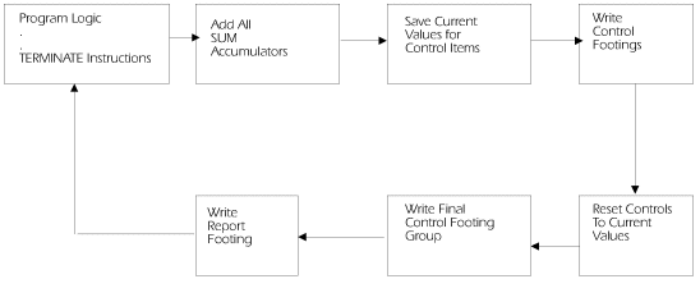


- The optional USE BEFORE REPORTING statement lets you specify any additional processing you want done to a heading or footing report

group, such as an additional calculation or a line edit, prior to printing.

- The TERMINATE statement completes the processing of a report. It is analogous to the CLOSE statement for individual files. *TERMINATE Logic Processing* below illustrates TERMINATE processing.

Figure 11-4. TERMINATE Logic Processing



Use special counters

APS Report Writer provides two special counters that you can use in any Procedure Division statement:

- You can reference LINE-COUNTER to determine when to print a PAGE HEADING or a PAGE FOOTING report group. The maximum value of the LINE-COUNTER is based on the number of lines per page specified in the PAGE LIMIT(S) clause.
- You can reference PAGE-COUNTER in a SOURCE statement to print the page number.

View sample reports

Sample Report Program Structure shows a sample report program structure. *Sample Program* shows a report mock-up, complete APS program code, generated COBOL source code, and printed report.

Sample Report Program Structure

INPUT-OUTPUT SECTION

```
KYWD  12-*----20---*----30---*----40---*----50---*--
IO    SELECT statement
```

FILE SECTION

```
KYWD  12-*----20---*----30---*----40---*----50---*--
FD    inputfile FD clauses
```

```
01    recordname       PIC picclause
FD    reportfile FD clauses
      REPORT IS | REPORTS ARE clause
```

REPORT SECTION

```
KYWD  12-*----20---*----30---*----40---*----50---*--
RED   reportfilename
      CODE clause
      CONTROL clause
      WRITE ROUTINE clause
      PAGE LIMIT clause
      FIRST DETAIL clause
      LAST DETAIL clause
      FOOTING clause

MOCK  mockupreportname
```

Report Group Types:

- Header Types (Report, Page, and Control Headers)

```
KYWD  12-*----20---*----30---*----40---*----50---*--
01    TYPE clause for report, page or control header
      MOCKUP LINES clause
      SOURCE clause | VALUE clause
```

- Detail Line Type

```
KYWD  12-*----20---*----30---*----40---*----50---*--
01    TYPE DETAIL
      MOCKUP LINES clause
      SOURCE clause | VALUE clause
      REFERENCE clause
```

- Footer Types (Report, Page, and Control Footers)

```
KYWD  12-*----20---*----30---*----40---*----50---*--
01    TYPE clause for report, page or control header
      MOCKUP clause
      SOURCE clause | VALUE clause
      SUM clause
```

PROCEDURE DIVISION

```
KYWD  12-*----20---*----30---*----40---*----50---*--
NTRY |
PROC
      .
      .
```

```
      .
      INITIATE statement
      .
      .
      .
      GENERATE statement
      .
      .
      .
      TERMINATE statement
      .
      .
      .
```

Sample Report Program Code

```
-KYWD- 12-*----20---*----30---*----40---*----50---*-
 IO    INPUT-FILE ASSIGN TO UT-S-FILEIN.
 IO    REPORT-OUTPUT-FILE ASSIGN TO UT-S-REPTOUT.

 FD    INPUT-FILE
       LABEL RECORDS ARE STANDARD
       BLOCK CONTAINS 0 CHARACTERS.
 01    PART-STOCK-REC                 PIC X(80).

 FD    REPORT-OUTPUT-FILE
       LABEL RECORDS ARE STANDARD
       REPORT IS STOCK-REPORT.
 RED   STOCK-REPORT
       CONTROLS ARE FINAL WS-LOCATION-CODE
       PAGE LIMIT IS 50
       FIRST DETAIL 10
       LAST DETAIL 40
       FOOTING 47.

 MOCK  STCKRPT

 01    TYPE IS REPORT HEADING NEXT GROUP NEXT PAGE
           LINE 20.
       MOCKUP LINES 1 THRU 6
       SOURCE WS-DATE

 01    TYPE IS PAGE HEADING.
       MOCKUP LINES 7 THRU 9
       SOURCE WS-DATE
       .
       .
       .
```

```
NTRY
      OPEN INPUT INPUT-FILE
      ...OUTPUT REPORT-OUTPUT-FILE
      ACCEPT WS-DATE-HOLD FROM DATE
      .
      .
      .
```

Painting Report Mock-Ups

To paint report mock-ups, perform the following steps:

1 **Access the Report Painter**
To access the Report Painter to create a new report mock-up or edit a current mock-up, do one of the following:

- From the Application Painter:
  - Enter a report mock-up name in the Reports field associated with your program.
  - Enter s in the selection field next to the name and press Enter.
- From the APS Painter Menu:
  - To edit a report mock-up, type rp in the Type field and the report mock-up name in the Member field. Then press Enter.
  - To browse a report mock-up, type b in the Command field, rp in the Type field, and the mock-up name in the Member field. Then press Enter.
  - To display a member list from which to select mock-ups, type rp in the Type field, leave the Member field blank, and press Enter. Then select the member from the member list.

Paint the mock-up

2 To create a mock-up, type the literals and output fields in columns 1 through 247, as follows.

- To create a literal field, type the literal characters.

- To create an output field, type a PIC character string, for example Z,ZZZ,ZZ9 or XXXXX. You can use the following COBOL edit masks:
  - Floating numeric formats, including: * $ + - S 9 Z
  - Alphanumeric masks

Figure 11-5. Creating a Prgram Mock-Up

```
EDIT --- REPORT: REPORT ------------------------------------ COLUMNS 001 0
COMMAND ===>                                                SCROLL ===> CSR
****** ************************* TOP OF DATA ***************************
000100                  WONDERFUL WIDGETS INCORPORATED
000200
000300                           STOCK REPORT
000400
000500                            XXXXXX XXX
000600
000700                           MID ATLANTIC
000800                           STOCK REPORT
000900                            XXXXXX XXX
001000
001100 LOCATION          LAST COUNT     QUANTITY IN    QUANTITY     QUANTITY
001200                     DATE           STOCK         ISSUED      RECEIVED
001300
001400 XXXXXXXXXXXX       99/99/99         ZZZ,ZZ9       ZZZ,ZZ9      ZZZ,ZZ9
001500
001600                                    _________     _________    _________
001700 TOTAL BY LOCATION:                 Z,ZZZ,ZZ9     Z,ZZZ,ZZ9    Z,ZZZ,ZZ9
001800
001900 TOTAL NUMBER OF SALES BY LOCATION:      ZZZ,ZZ9
****** ************************ BOTTOM OF DATA *************************
```

3 Use ISPF editor commands to edit the text. To view a mock-up larger than your screen, press F10 and F11 to scroll left and right.

Special Considerations

- When you add or delete a line from your report mock-up, always renumber the line numbers, so you can accurately reflect them in your report program code. To do so, type ren or renum in the Command line and press Enter.
- The first character of the report mock-up name must be alphabetic, @, or #; the remaining characters can be alphabetic, numeric, @, $, or #.
- Reports can be a maximum of 200 lines and 247 columns.
- In your program, the data field PIC strings on the mock-up must match the detail line data item descriptions in your Report Section.

Report Writer matches the mock-up fields to the program field descriptions from left to right, from top to bottom.

- The report mock-up determines the columns where literals and data items print. Your report program code determines the lines where they print.
- APS stores report mock-ups in your APS Project Group APSREPT data set.

Creating Report Programs

To create a batch report program in the Program Painter, follow these steps:

Specify DB and DC targets

1 To specify your DB and DC targets in the Application Painter, follow steps 2 and 3 in *Creating Batch Programs*.

Access the program

2 To create or edit a program, do one of the following:

- From the Application Painter, type the program name in the Program field on the same line as its associated mock-up, and type s in the selection field next to the program name. Then press Enter.
- From the APS Painter Menu, type the program name in the Member field and press Enter. Or, leave the Member field blank, press Enter, and then select the applicable member from the member list.

Code the input/output statements

3 Specify the FILE-CONTROL SELECT information with the Program Painter IO keyword. Use the following format:

```
-KYWD- 12-*----20---*----30---*----40---*---50
 IO    filename ASSIGN [TO] systemname
       Applicable COBOL FILE-CONTROL clauses
```

4 Code the program input file description with the Program Painter FD keyword, as follows:

```
-KYWD- 12-*----20---*----30---*----40---*----
 FD    inputfilename
       LABEL RECORDS clause
       BLOCK CONTAINS clause
```

```
       [Other applicable COBOL FD clauses]
 01    inputrecordname              PIC clause.
```

For example:

```
-KYWD- 12-*----20---*----30---*----40---*----
 FD    INPUT-FILE
       LABEL RECORDS ARE STANDARD
       BLOCK CONTAINS 0 CHARACTERS
 01    PART-STOC-REC              PIC X(80).
```

5 Determine the output record size. The default size is 133, the standard mock-up size of 132 plus 1 byte for the carriage control character. To define a different report record size, calculate the size as follows:

Record Size = Report mock-up size (maximum 247 characters)
+ 1 byte for carriage control
+ 2 bytes for the CODE clause, if used.

6 Then, code the program output file description with the Program Painter FD keyword, as follows. To accept the default record size, omit the RECORD CONTAINS clause.

```
-KYWD- 12-*----20---*----30---*----40---*----
 FD    outputfilename
       LABEL RECORDS clause
       [RECORD CONTAINS clause]
       [Other applicable COBOL FD clauses]
       REPORT IS|ARE reportname1 [... reportname15]
 01    outputrecordname              PIC clause.
```

For example:

```
-KYWD- 12-*----20---*----30---*----40---*----
 FD    REPORT-OUTPUT-FILE
       LABEL RECORDS ARE STANDARD
       REPORT IS STOCK-REPORT

 WS    WS-PART-STOC-REC.
       05  WS-LOCATION-CODE       PIC X(12)  VALUE SPACES.
       05  WS-LAST-COUNT-DATE.
           10  WS-LAST-COUNT-MONTH  PIC 99     VALUE 0.
           10  WS-LAST-COUNT-DAY    PIC 99     VALUE 0.
           10  WS-LAST-COUNT-YEAR   PIC 99     VALUE 0.
       05  WS-QTY-IN-STOCK          PIC 9(6)   VALUE 0.
       05  WS-QTY-ISSUED            PIC 9(6)   VALUE 0.
       05  WS-QTY-RECEIVED          PIC 9(6)   VALUE 0.
```

```
       05  WS-NO-OF-SALES          PIC 9(6)   VALUE 0.
       05  FILLER                PIC X(40)  VALUE SPACES.
```

Add Working-Storage entries

7 After the output record description, add any Working-Storage entries needed for your report at this point in your program. For example:

```
-KYWD- 12-*----20---*----30---*----40---*----
 WS    WS-DATE.
       05  WS-DATE-MM             PIC 9(2).
       05  WS-DATE-DD             PIC 9(2).
       05  WS-DATE-YY             PIC 9(2).
 WS    WS-DATE-HOLD.
       05  WS-DATE-YY-X           PIC 9(2).
       05  WS-DATE-MM-X           PIC 9(2).
       05  WS-DATE-DD-X           PIC 9(2).
```

Identify the report

8 To identify the report, code the RED keyword and the report name, as follows:

```
-KYWD- 12-*----20---*----30---*----40---*----
 RED   reportname
```

For example:

```
-KYWD- 12-*----20---*----30---*----40---*----
 RED   STOCK-REPORT
```

Describe the printed page

9 Identify the data items to test for a control break. The order in which you code the data items creates the control hierarchy, where FINAL is the highest control, the first data item is the major control, and the last data item is the minor (lowest) control. Use the following syntax:

```
-KYWD- 12-*----20---*----30---*----40---*----
 RED   reportname
       [CONTROL [IS] | CONTROLS [ARE] [FINAL]
           dataname1 ... [datanameN]]
```

For example:

```
-KYWD- 12-*----20---*----30---*----40---*----
 RED   STOCK-REPORT
       CONTROLS ARE FINAL WS-LOCATION-CODE
```

10 Optionally specify the length and vertical subdivisions of the printed page, as follows:

a Code the number of printable lines on each page in the PAGE LIMIT option.

b Code the line number where you want the first control heading line or detail line of the report body to print on each page in the FIRST DETAIL option. Remember to leave space at the top of the page for any report heading and page heading lines when calculating the first detail line number.

c Code the line number where you want the last detail line of the report body to print on each page in the LAST DETAIL option. Remember to leave space at the bottom of the page for any control break lines, page footing lines, and report footing lines when calculating the last detail line number.

d Code the line number where you want the last control footing line to print for each page in the FOOTING option. Remember to leave space at the bottom of the page for any page footing and report footing lines when calculating the last control footing number.

Use the following syntax:

```
-KYWD- 12-*----20---*----30---*----40---*----
 RED   reportname
       [CONTROL [IS] | CONTROLS [ARE] [FINAL]
           dataname1 ... [datanameN]]
       [PAGE [LIMIT IS | LIMITS ARE] number
[LINE|LINES]
           [FIRST DETAIL firstlinenumber]
           [LAST DETAIL lastlinenumber]
           [FOOTING footinglinenumber]].
```

For example:

```
-KYWD- 12-*----20---*----30---*----40---*----
 RED   STOCK-REPORT
       CONTROLS ARE FINAL WS-LOCATION-CODE
       PAGE LIMIT IS 50
       FIRST DETAIL 10
       LAST DETAIL 40
       FOOTING 47.
```

Identify the mock-up

11 Specify the mock-up named in the Application Painter and painted in the Report Painter with the MOCK keyword, as follows:

```
-KYWD- 12-*----20---*----30---*----40---*----
 MOCK  reportmockupname
```

For example:

```
-KYWD- 12-*----20---*----30---*----40---*----
 MOCK  STOCKRPT
```

Identify each report group

12 Identify each report group with the Program Painter 01 keyword and the TYPE clause, as follows:

a Code an 01 TYPE statement for each report group. Note that:

- You must assign an identifying data name for the DETAIL report group, the report body. Identifiers for the other report groups are optional. You use these identifying data names in the Procedure Division to refer to the various report groups.
- The TYPE statements for CONTROL HEADING and CONTROL FOOTING report groups must indicate the name of control break field that causes the control break. This control data name must correspond to a data item specified in the CONTROLS option of the RED statement.

b For each report group, you can optionally designate the line number where the first line of the report group prints, using the LINE option.

c For each report group, you can optionally designate the line number where the first line of the next report group prints, using the NEXT GROUP option.

Use the following syntax:

```
-KYWD- 12-*----20---*----30---*----40---*----
                          PAGE HEADING
                          PAGE FOOTING
                          REPORT HEADING
 01  [identifier] TYPE IS REPORT FOOTING
                          CONTROL HEADING [FINAL]|controlname
                          CONTROL FOOTING [FINAL]|controlname
                          DETAIL
                          number
        [LINE [NUMBER IS] PLUS number
                           NEXT PAGE       ]
```

```
                    number
  [NEXT GROUP [IS] PLUS number
                   NEXT PAGE       ]
```

For example:

```
-KYWD- 12-*----20---*----30---*----40---*----
 MOCK  STOCKRPT
 01    TYPE IS REPORT HEADING
           NEXT GROUP IS NEXT PAGE
 01    TYPE IS PAGE HEADING
           NEXT GROUP PLUS 3
 01    TYPE IS CONTROL HEADING WS-LOCATION-CODE
           NEXT GROUP PLUS 1
 01    PART-DETAIL TYPE IS DETAIL
           NEXT GROUP PLUS 1
 01    TYPE IS CONTROL FOOTING WS-LOCATION-CODE
 01    TYPE IS CONTROL FOOTING FINAL
 01    TYPE IS PAGE FOOTING
 01    TYPE IS REPORT FOOTING
           LINE PLUS 2
```

Map report lines to the mock-up

13 To map the each report group to the mock-up, follow each TYPE clause with a MOCKUP clause, as follows:

```
-KYWD- 12-*----20---*----30---*----40---*----
       MOCKUP LINE[S] linenumber1 [THRU linenumberN]
```

For example, the following code:

```
-KYWD- 12-*----20---*----30---*----40---*----
 MOCK  STOCKRPT
 01    TYPE IS REPORT HEADING
           NEXT GROUP IS NEXT PAGE
       MOCKUP LINES 1 THRU 5
 01    TYPE IS PAGE HEADING
           NEXT GROUP PLUS 2
       MOCKUP LINES 6 THRU 8
 01    TYPE IS CONTROL HEADING WS-LOCATION-CODE
       MOCKUP LINES 10 THRU 12
 01    PART-DETAIL TYPE IS DETAIL
           NEXT GROUP PLUS 1
       MOCKUP LINE 13
 01    TYPE IS CONTROL FOOTING WS-LOCATION-CODE
       MOCKUP LINES 14 THRU 19
 01    TYPE IS CONTROL FOOTING FINAL
       MOCKUP LINES 20 THRU 24
 01    TYPE IS PAGE FOOTING
           LINE PLUS 2
```

```
                    MOCKUP LINE 25
                01  TYPE IS REPORT FOOTING
                        LINE PLUS 2
                    MOCKUP LINE 27
```

Corresponds to the following mock-up:

```
****** ***************************TOP OF DATA************************
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6---
000100                  WONDERFUL WIDGETS INCORPORATED
000200                          MID-ATLANTIC
000300                           STOCK REPORT
000400                            XXXXXX XXX
000500
000600                          MID-ATLANTIC
000700                          STOCK REPORT
000800                          XXXXXX XXX
000900
001000 LOCATION       LAST COUNT   QUANTITY   QUANTITY   QUANTITY
001100                   DATE      IN STOCK    ISSUED    RECEIVED
001200
001300 XXXXXXXXXXXX    99/99/99     ZZZ,ZZ9     ZZ,ZZ9    ZZZ,ZZ9
001400
001500                             ---------  ---------  ---------
001600 TOTAL BY LOCATION:          Z,ZZZ,ZZ9   Z,ZZZ,ZZ9  Z,ZZZ,ZZ9
001700
001800 TOTAL NUMBER OF SALES BY LOCATION:       ZZZ,ZZ9
001900
002000 TOTAL WONDERFUL WIDGETS IN STOCK:      Z,ZZZ,ZZ9
002100 TOTAL WONDERFUL WIDGETS ISSUED:        Z,ZZZ,ZZ9
002200 TOTAL WONDERFUL WIDGETS RECEIVED:      Z,ZZZ,ZZ9
002300 TOTAL WONDERFUL WIDGETS SOLD:          Z,ZZZ,ZZ9
002400
002500                               PAGE ZZZ9
002600
002700                       *****  END OF REPORT  *****
002800
002900
```

Map report fields to the mock-up

14 For the applicable report groups, indicate which data items supply values to output fields with the SOURCE statement. Use the following guidelines:

- Map the data items within the report group from left to right, top to bottom, as they will appear on the printed report.
- Include picture clauses if multiple fields are strung together on the mock-up. A PIC clause indicates the next matching COBOL

picture in the mock-up is the COBOL picture for this data item. See also *Special Considerations*.

- Optionally print spaces when the value of the field is zero by specifying the BLANK WHEN ZERO option.
- Optionally justify the field value with the JUSTIFIED RIGHT option.
- Optionally print the value of the field only when it changes value with the CHANGE INDICATE option.
- Optionally print the value of the field only on the first occurrence of the report group after a control break or a page advance with the GROUP INDICATE option.

Use the following syntax:

```
-KYWD- 12-*----20---*----30---*----40---*----
       SOURCE [IS] dataname [PIC picclause]
           [BLANK [WHEN] ZERO]
           [JUSTIFIED|JUST [RIGHT]
           [CHANGE INDICATE|GROUP INDICATE]
```

For example:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
 01    TYPE IS REPORT HEADING
           NEXT GROUP IS NEXT PAGE
       MOCKUP LINES 1 THRU 5
       SOURCE IS WS-DATE
 01    TYPE IS PAGE HEADING
           NEXT GROUP PLUS 3
       MOCKUP LINES 6 THRU 8
       SOURCE IS WS-DATE
 01    PART-DETAIL TYPE IS DETAIL
           NEXT GROUP PLUS 1
       MOCKUP LINE 13
       SOURCE WS-LOCATION-CODE    GROUP INDICATE
       SOURCE WS-LAST-COUNT-MONTH
       SOURCE WS-LAST-COUNT-DAY
       SOURCE WS-LAST-COUNT-YEAR
       SOURCE WS-QTY-IN-STOCK
       SOURCE WS-QTY-ISSUED
       SOURCE WS-QTY-RECEIVED
 01    TYPE IS PAGE FOOTING
           LINE PLUS 2
       MOCKUP LINE 24
       SOURCE IS PAGE-COUNTER
```

15 Define any non-printing detail items that you want Report Writer to sum and total for control breaks, such as an employee count or the number of sales in a given location, with the REFERENCE statement. A REFERENCE field value never displays when the detail line prints. If you code a corresponding SUM statement (see the next step), APS adds the field value to an internal sum accumulator.

Name the data item and define its format, as follows:

```
REFERENCE [IS] dataname PIC picclause
```

For example:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
 01    PART-DETAIL TYPE IS DETAIL
           NEXT GROUP PLUS 1
       MOCKUP LINE 13
       SOURCE WS-LOCATION-CODE    GROUP INDICATE
       SOURCE WS-LAST-COUNT-MONTH
       SOURCE WS-LAST-COUNT-DAY
       SOURCE WS-LAST-COUNT-YEAR
       SOURCE WS-QTY-IN-STOCK
       SOURCE WS-QTY-ISSUED
       SOURCE WS-QTY-RECEIVED
       REFERENCE WS-NO-OF-SALES   PIC 9999
```

See also *Special Considerations*.

16 For the CONTROL FOOTING report groups, sum the data items, previously identified with a SOURCE or REFERENCE statement, for control breaks with the SUM statement. Use the following guidelines:

- Sum the data items within the report group from left to right, top to bottom, as they will appear on the printed report.
- If the report has more than one detail line, use the UPON option to name the line where the summing takes place.
- To override the APS default of resetting SUM accumulators to zero after each control break, use the RESET option to do a running total and specify which control break should reset the accumulator.
- To automatically move the value in the Report Writer field accumulator to a field you can reference in the Procedure Division, for example, to test the field or do further calculations,

create the field by naming it in the DATA-NAME *fieldname* option.

- Include a picture clause for SUM statements if multiple fields are strung together on the mock-up. A PIC clause indicates the next matching COBOL picture in the mock-up is the COBOL picture for this data item. See also *Special Considerations*.

Use the following syntax:

```
SUM dataname [dataname] ...
    [UPON detlineidentifier [detlineidentifier] ...]
    [RESET [FINAL] controlname]
    [DATA-NAME fieldname]
    [PICTURE picclause]
```

For example:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
 MOCK  STOCKRPT

 01    TYPE IS REPORT HEADING
           NEXT GROUP IS NEXT PAGE
       MOCKUP LINES 1 THRU 5
       SOURCE WS-DATE

 01    TYPE IS PAGE HEADING
           NEXT GROUP PLUS 3
       MOCKUP LINES 6 THRU 8
       SOURCE WS-DATE

 01    TYPE IS CONTROL HEADING WS-LOCATION-CODE
       MOCKUP LINES 10 THRU 12

 01    PART-DETAIL TYPE IS DETAIL
           NEXT GROUP PLUS 1
       MOCKUP LINE 13
       SOURCE WS-LOCATION-CODE    GROUP INDICATE
       SOURCE WS-LAST-COUNT-MONTH
       SOURCE WS-LAST-COUNT-DAY
       SOURCE WS-LAST-COUNT-YEAR
       SOURCE WS-QTY-IN-STOCK
       SOURCE WS-QTY-ISSUED
       SOURCE WS-QTY-RECEIVED
       REFERENCE WS-NO-OF-SALES   PIC 9999

 01    TYPE IS CONTROL FOOTING WS-LOCATION-CODE
       MOCKUP LINES 14 THRU 18
       MOCKUP LINES 19 THRU 22
```

```
       SUM WS-QTY-IN-STOCK
       SUM WS-QTY-ISSUED
       SUM WS-QTY-RECEIVED
       SUM WS-NO-OF-SALES

 01    TYPE IS CONTROL FOOTING FINAL
       MOCKUP LINES 19 THRU 22
       SUM WS-QTY-IN-STOCK
       SUM WS-QTY-ISSUED
       SUM WS-QTY-RECEIVED
       SUM WS-NO-OF-SALES

 01    TYPE IS PAGE FOOTING
           LINE PLUS 2
       MOCKUP LINE 24
       SOURCE IS PAGE-COUNTER

 01    TYPE IS REPORT FOOTING
           LINE PLUS 3
       MOCKUP LINE 26
```

Code the report logic

17 Code the Procedure Division logic under the NTRY (or PROC) keyword to produce and generate the report, as follows:

a Initialize all report counters and set up control heading and footing items with the INITIATE statement.

b Process the detail lines with a GENERATE statement. If the report has multiple detail lines, code multiple GENERATE statements.

c End report processing with the TERMINATE statement.

Use the following structure:

```
-KYWD- 12-*----20---*----30---*----40---*----50---
 NTRY
            .
            .
       OPEN INPUT filename1
       ... OUTPUT filename2
            .
            .
       INITIATE reportname
            .
            .
       IF ..............
           GENERATE detlineidentifier
            .
```

```
        .
    TERMINATE reportname
        .
        .
    CLOSE filename1 filename2
```

For example:

```
NTRY
     OPEN INPUT INPUT-FILE
     ...OUTPUT REPORT-OUTPUT-FILE
     ACCEPT WS-DATE-HOLD FROM DATE
     MOVE WS-DATE-YY-X TO WS-DATE-YY
     MOVE WS-DATE-MM-X TO WS-DATE-MM
     MOVE WS-DATE-DD-X TO WS-DATE-DD
     INITIATE STOCK-REPORT
     REPEAT
         READ INPUT-FILE INTO WS-PART-STOCK-REC
     UNTIL AT END ON INPUT-FILE
         GENERATE DETAIL-LINE
     TERMINATE STOCK-REPORT
     CLOSE INPUT-FILE
     ... REPORT-OUTPUT-FILE
```

18 Repeat steps 7 through 17 for each report named in the File Section REPORT clause.

Special Considerations

- If you define a report record with the WRITE ROUTINE clause, the default record size is 248. If your RED keyword statement includes the CODE clause, the default value is 250. For more information, see the APS Reference.
- Report Writer treats each report group specified in the TYPE clause as a unit and always prints the entire group on one page--it never begins the group on one page and completes it on another.
- You can add Working-Storage entries before the I/O description at the beginning of the program, after the output record description, after an 01 TYPE statements, and after yout Procedure Division code.
- When identifing controls in the CONTROL clause, *dataname* must be an elementary data name. In the following example, B cannot be used as a control variable because it is a group data item. To make B

into an elementary data item, use the REDEFINES clause as shown below:

```
WS01  A                          PIC X(2).
WS01  B.
      02  B-1                    PIC 9(4).
      02  B-2                    PIC 9(4).
WS01  B-REDEF  REDEFINES B       PIC X(8).
   .
   .
   .
RED TEST-REPORT
    CONTROLS ARE A B-REDEF
```

- APS creates an internal SUM accumulator field for each data item specified. The name of this field is *dataname-nnnn*, where *nnnn* is a 4-digit number. Each time the detail line containing the data item prints, APS adds its value to the accumulator. APS clears the accumulators either after each control break (the default) or after a control break you specifiy with the RESET option.

- If a data item contains a PIC clause in a SOURCE or SUM statement, it indicates that the next matching COBOL picture in the mock-up is the COBOL picture for the statement. APS compares it with an equal number of the next unassigned characters. If no match occurs, the comparison moves one position to the right until a match is found. If no match is found, an error message is printed. For example:

  Report mock-up:

```
000001              A TEST PROGRAM
000002
000003      FIELD-1      FIELD-2      FIELD-3
000004      XXXX         XXXX         XXXX
```

  Report program:

```
-KYWD- 12-*----20---*----30---*----40---*----
 01    DET-LINE TYPE DETAIL LINE.
       MOCKUP LINE 4
       SOURCE DATA-1
       SOURCE DATA-2
       SOURCE DATA-3     PIC X(2)
       SOURCE DATA-4
```

  Report Writer matches DATA-1 and DATA-2 directly to the mock-up. It then matches the PIC clause in the SOURCE statement for DATA-3 to the first two X characters under FIELD-3, and the two remaining

X characters to DATA-4. If you omit the PIC clause on the DATA-3 SOURCE statement, an error occurs. If you omit the DATA-4 SOURCE statement, APS considers the XX a literal, because there are no source statements remaining.

- When you code a REFERENCE statement, the PIC clause must match the PIC clause in the record description. For example:

```
-KYWD- 12-*----20---*----30---*----40---*----
 01    COST-DETAIL TYPE DETAIL
       MOCKUP LINE 9
       SOURCE WS-DEPT
       SOURCE WS-EMPLOYEE
       SOURCE WS-CITY
       REFERENCE EMP-CTR   PIC 999
 01    TYPE CONTROL FOOTING
       MOCKUP LINE 9
       SOURCE WS-DEPT
       SUM EMP-CTR
 WS01  EMP-CTR     PIC 999   VALUE 1.
```

  If one of the PIC clauses were PIC 9(3), Report Writer would not find a match.

- In a REFERENCE statement, the data item referenced must be defined in Working-Storage with a VALUE clause. The value in the VALUE clause tells Report Writer the increment to add to the internal accumulator each time the detail line prints. In the previous example, APS adds 1 to the internal accumulator whenever the detail line prints.

Generate Multiple SUM or SOURCE Statements

Instead of coding individual statements to source or sum sequential suffixed data items or array elements, APS provides an iterative expression feature that lets you code only one SOURCE or SUM statement, which generates multiple statements.

Suffixed Data Elements

The iterative expression syntax for suffixed data elements is:

```
dataitem-#startnum[/endnum[/incnum]]
```

where the pound sign (#) indicates the starting number of an iteration; *startnum* and *endumn* indicate the range of the iteration. The slash (/) between them generates a THRU. The *incnum* is the number by which the iteration is incremented (default is 1); its leading slash generates a BY. For example:

```
SOURCE MONTH-#1/6 BLANK WHEN ZERO
```

generates an iteration of six SOURCE statements, suffixed numerically, 1 through 6, for example, MONTH-1 through MONTH-6. If you specify only one number, the iteration assumes the starting number to be 1. For example, SOURCE MONTH-#6 is equivalent to SOURCE MONTH-#1/6.

Without the iterative expression, you would code the above statement as follows:

```
SOURCE MONTH-1 BLANK WHEN ZERO
SOURCE MONTH-2 BLANK WHEN ZERO
SOURCE MONTH-3 BLANK WHEN ZERO
SOURCE MONTH-4 BLANK WHEN ZERO
SOURCE MONTH-5 BLANK WHEN ZERO
SOURCE MONTH-6 BLANK WHEN ZERO
```

In the following example, the four SUM statements:

```
SUM MONTH-9-DATA
SUM MONTH-10-DATA
SUM MONTH-11-DATA
SUM MONTH-12-DATA
```

can instead be coded as:

```
SUM MONTH-#9/12-DATA
```

The following example increments an iteration of SOURCE statements by 2 instead of the default 1:

```
SOURCE ELEMENT-#6/12/2
```

which generates:

```
SOURCE ELEMENT-6
```

```
SOURCE ELEMENT-8
SOURCE ELEMENT-10
SOURCE ELEMENT-12
```

The PIC clause for each SUM or SOURCE statement generated by the iterative expression is taken from the mock-up. APS matches the mock-up and detail line data item descriptions as the increment statements generate.

Array Items

The iterative expression syntax for a complex array is:

```
arrayitem (#startnum1[/endnum1[/incnum1]][])
              [,#startnum2[/endnum2[/incnum2]]][)]
                  [,#startnum3[/endnum3[/incnum3]]])
```

Parentheses indicate an array. Use up to three # symbols to indicate three dimensions of an array. Separate the subscript ranges with commas. All of the symbols used for generating suffixed data items, above, apply to each array range.

Each dimension is described by a separate Data Division entry with an OCCURS clause.

The SOURCE statements in the following example, which reference a 2 by 3 array:

```
SOURCE ARRAY-ITEM (1, 1)
SOURCE ARRAY-ITEM (1, 2)
SOURCE ARRAY-ITEM (1, 3)
SOURCE ARRAY-ITEM (2, 1)
SOURCE ARRAY-ITEM (2, 2)
SOURCE ARRAY-ITEM (2, 3)
```

can, instead, be coded as:

```
SOURCE ARRAY-ITEM (#2, #3)
```

The following example produces SUM statements for each element of a three dimensional array of 3 by 2 by 3.

```
SUM TABLE ELEMENT (#3, #2, #3)
```

The following iterative expressions are examples of ranges within a one dimensional array:

```
SUM EXT-SALES-DOLLARS (#1/3)        PIC Z,ZZ9
     SUM QTR-1-SALES-DOLLARS        PIC ZZ,ZZ9
     SUM EXT-SALES-DOLLARS (#4/6)   PIC Z,ZZ9
     SUM QTR-2-SALES-DOLLARS        PIC ZZ,ZZ9
     SUM EXT-SALES-DOLLARS (#7/9)   PIC Z,ZZ9
     SUM QTR-3-SALES-DOLLARS        PIC ZZ,ZZ9
     SUM EXT-SALES-DOLLARS (#10/12) PIC Z,ZZ9
     SUM QTR-4-SALES-DOLLARS        PIC ZZ,ZZ9
     SUM YR-SALES-DOLLARS           PIC $$$$,$$$
```

Mapping Considerations

The sequence in which mock-up fields are matched with the data item descriptions is the same as a page of text is read--from left to right across each line of the mock-up starting with the top line and continuing to the bottom.

APS matches the report mock-up fields to the data item description entries in your program according to the following rules:

- APS considers the following to be literals:
  - The COBOL picture character A.
  - Any one or more consecutive non-space, non-picture characters.
  - Any single COBOL picture character, that is preceded and followed by a space. Exception to this rule: 9 and X.
  - A string of hyphens because of its frequent use for underlining.
- APS considers a single COBOL picture character, such as -, X, Z, or 9, that is embedded in a string of non-blank, non-picture characters as part of a literal. For example, the following are literals:

  ```
  1979
  WXYZ
  EXTRA
  WIZARD
  ```

and the following are pictures beside literals:

```
#99     Literal is #, PIC is 99.
l999    Literal is l, PIC is 999.
Section-999  Literal is SECTION, PIC is -999.
```

- APS considers any legal COBOL picture longer than one character to be a COBOL picture, except for the letter S and the hyphen (-), and matches it to the next data item description in the program.
- APS generates a VALUE statement for each literal in the mockup, and does not match the literal with the data item descriptions in the program.
- APS considers any consecutive PIC characters in the mock-up as one PIC character string, unless the string is matched with PIC clauses in multiple, consecutive SOURCE statements.
- APS assigns each PIC character string as the PIC for the next sequential data item description, unless the next description contains a PIC clause.
- When a data item description contains a PIC clause, APS compares it with an equal number of characters in the mock-up, starting with the next sequential, unassigned character in the mock-up.
- When comparing a data item description entry with a PIC clause to an equal number of characters in the mock-up and a non-match occurs, APS continues the comparison by moving one position to the right until it finds a match. APS considers the non-matched characters from this process to be a literal, and generates a VALUE entry that precedes the data item description with the PIC clause that initiated the comparison.
- When a PIC clause in a data item description does not match any series of mock-up characters from the start of a comparison to the end of the mock-up, APS terminates processing and generates an error message.

Sample Program

This topic includes the report mock-up, complete program, the generated source code, and the final printed report for the sample program illustrated in the procedure for *Creating Report Programs*.

Program Painter source

```
KYWD 12--+--20--+---+-30+---+---40--+---+-50+---+---
 IO  INPUT-FILE ASSIGN TO UT-S-FILEIN.
 IO  REPORT-OUTPUT-FILE ASSIGN TO UT-S-REPTOUT.

 FD  INPUT-FILE
     LABEL RECORDS ARE STANDARD
     BLOCK CONTAINS 0 CHARACTERS.
 01  PART-STOCK-REC              PIC X(80).

 FD  REPORT-OUTPUT-FILE
     LABEL RECORDS ARE STANDARD
     REPORT IS STOCK-REPORT.

 WS  WS-PART-STOCK-REC.
     05  WS-LOCATION-CODE
         ... PIC X(12)  VALUE SPACES.
     05  WS-LAST-COUNT-DATE.
         10  WS-LAST-COUNT-MONTH PIC 99    VALUE 0.
         10  WS-LAST-COUNT-DAY   PIC 99    VALUE 0.
         10  WS-LAST-COUNT-YEAR  PIC 99    VALUE 0.
     05  WS-QTY-IN-STOCK         PIC 9(6)  VALUE 0.
     05  WS-QTY-ISSUED           PIC 9(6)  VALUE 0.
     05  WS-QTY-RECEIVED         PIC 9(6)  VALUE 0.
     05  WS-NO-OF-SALES          PIC 9(4)  VALUE 0.
     05  FILLER
         ...PIC X(40) VALUE SPACES.
 WS  WS-DATE.
     05  WS-DATE-MM              PIC 9(2).
     05  WS-DATE-DD              PIC 9(2).
     05  WS-DATE-YY              PIC 9(2).
 WS  WS-DATE-HOLD.
     05  WS-DATE-YY-X            PIC 9(2).
     05  WS-DATE-MM-X            PIC 9(2).
     05  WS-DATE-DD-X            PIC 9(2).
 RED STOCK-REPORT
     CONTROLS ARE FINAL WS-LOCATION-CODE
     PAGE LIMIT IS 50
     FIRST DETAIL 10
     LAST DETAIL 40
     FOOTING 47.
```

```
MOCK STCKRPT
01  TYPE IS REPORT HEADING NEXT GROUP
        NEXT PAGE
    MOCKUP LINES 1 THRU 6
    SOURCE WS-DATE

01  TYPE IS PAGE HEADING.
    MOCKUP LINES 7 THRU 9
    SOURCE WS-DATE

01  TYPE IS CONTROL HEADING WS-LOCATION-CODE.
    MOCKUP LINES 10 THRU 13

01  DETAIL-LINE TYPE IS DETAIL.
    MOCKUP LINE 14
    SOURCE WS-LOCATION-CODE      GROUP INDICATE
    SOURCE WS-LAST-COUNT-MONTH  PIC 99
    SOURCE WS-LAST-COUNT-DAY    PIC 99
    SOURCE WS-LAST-COUNT-YEAR   PIC 99
    SOURCE WS-QTY-IN-STOCK
    SOURCE WS-QTY-ISSUED
    SOURCE WS-QTY-RECEIVED
    REFERENCE WS-NO-OF-SALES    PIC ZZZ9

01  TYPE IS CONTROL FOOTING WS-LOCATION-CODE.
    MOCKUP LINES 15 THRU 21
    SUM WS-QTY-IN-STOCK
    SUM WS-QTY-ISSUED
    SUM WS-QTY-RECEIVED
    SUM WS-NO-OF-SALES

01  TYPE IS CONTROL FOOTING FINAL.
    MOCKUP LINES 22 THRU 30
    SUM WS-QTY-IN-STOCK
    SUM WS-QTY-ISSUED
    SUM WS-QTY-RECEIVED
    SUM WS-NO-OF-SALES

01  TYPE IS PAGE FOOTING.
    MOCKUP LINE 31
    SOURCE PAGE-COUNTER

01  TYPE IS REPORT FOOTING LINE PLUS 2.
    MOCKUP LINES 32 THRU 33

NTRY
    OPEN INPUT INPUT-FILE
    ...OUTPUT REPORT-OUTPUT-FILE
```

```
                    ACCEPT WS-DATE-HOLD FROM DATE
                    MOVE WS-DATE-YY-X TO WS-DATE-YY
                    MOVE WS-DATE-MM-X TO WS-DATE-MM
                    MOVE WS-DATE-DD-X TO WS-DATE-DD
                    INITIATE STOCK-REPORT
                    REPEAT
                        READ INPUT-FILE INTO WS-PART-STOCK-REC
                    UNTIL AT END ON INPUT-FILE
                        GENERATE DETAIL-LINE
                    TERMINATE STOCK-REPORT
                    CLOSE INPUT-FILE
                    ... REPORT-OUTPUT-FILE
                ****************** BOTTOM OF DATA *****************
```

Generated source

```
%   &AP-GEN-VER = 3000
%   &AP-PGM-D = "STOCK1"
%   &AP-GEN-DC-TARGET = "MVS"
%   &AP-TP-ENTRY-KYWD-SEEN = 1
%   &AP-FILE-CONTROL-SEEN = 1

IDENTIFICATION DIVISION.
PROGRAM-ID.                     STOCK1.
AUTHOR.                         AP-SYSTEM GENERATED.
DATE-WRITTEN.                   910125.
DATE-COMPILED.                  &COMPILETIME.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER.                &SYSTEM.
OBJECT-COMPUTER.                &SYSTEM.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO UT-S-FILEIN.
    SELECT REPORT-OUTPUT-FILE ASSIGN TO UT-S-REPTOUT.

DATA DIVISION.

FILE SECTION.

FD  INPUT-FILE
    LABEL RECORDS ARE STANDARD
    BLOCK CONTAINS 0 CHARACTERS.
01  PART-STOCK-REC             PIC X(80).
```

```
FD  REPORT-OUTPUT-FILE
    LABEL RECORDS ARE STANDARD
    REPORT IS STOCK-REPORT.

WORKING-STORAGE SECTION.
01  WS-PART-STOCK-REC.
    05  WS-LOCATION-CODE        PIC X(12) VALUE SPACES.
    05  WS-LAST-COUNT-DATE.
        10  WS-LAST-COUNT-MONTH PIC 99    VALUE 0.
        10  WS-LAST-COUNT-DAY   PIC 99    VALUE 0.
        10  WS-LAST-COUNT-YEAR  PIC 99    VALUE 0.
    05  WS-QTY-IN-STOCK         PIC 9(6)  VALUE 0.
    05  WS-QTY-ISSUED           PIC 9(6)  VALUE 0.
    05  WS-QTY-RECEIVED         PIC 9(6)  VALUE 0.
    05  WS-NO-OF-SALES          PIC 9(4)  VALUE 0.
    05  FILLER                  PIC X(40) VALUE SPACES.
01  WS-DATE.
    05  WS-DATE-MM              PIC 9(2).
    05  WS-DATE-DD              PIC 9(2).
    05  WS-DATE-YY              PIC 9(2).
01  WS-DATE-HOLD.
    05  WS-DATE-YY-X            PIC 9(2).
    05  WS-DATE-MM-X            PIC 9(2).
    05  WS-DATE-DD-X            PIC 9(2).

REPORT SECTION.
RED STOCK-REPORT
    CONTROLS ARE FINAL, WS-LOCATION-CODE
    PAGE LIMIT IS 50
    FIRST DETAIL 10
    LAST DETAIL 40
    FOOTING 47.

                WONDERFUL WIDGETS INCORPORATED

                        STOCK REPORT

                         XXXXXX XXX

                        MID-ATLANTIC
                        STOCK REPORT
                         XXXXXX XXX

LOCATION          LAST COUNT     QUANTITY IN    QUANTITY    QUANTITY
                     DATE          STOCK         ISSUED     RECEIVED

XXXXXXXXXXXX       99/99/99        ZZZ,ZZ9      ZZZ,ZZ9     ZZZ,ZZ9
```

```
                                    ---------     ---------  ---------
TOTAL BY LOCATION:                  Z,ZZZ,ZZ9     Z,ZZZ,ZZ9  Z,ZZZ,ZZ9

TOTAL NUMBER OF SALES BY LOCATION:       ZZZ,ZZ9

TOTAL WONDERFUL WIDGETS IN STOCK:      Z,ZZZ,ZZ9

TOTAL WONDERFUL WIDGETS ISSUED:        Z,ZZZ,ZZ9

TOTAL WONDERFUL WIDGETS RECEIVED:      Z,ZZZ,ZZ9

TOTAL WONDERFUL WIDGETS SOLD:          Z,ZZZ,ZZ9

                                                          PAGE ZZZ9
                               *****  END OF REPORT  *****

01  TYPE IS REPORT HEADING NEXT GROUP NEXT PAGE.
MOCKUP LINES 1 THRU 6
SOURCE WS-DATE

01  TYPE IS PAGE HEADING.
MOCKUP LINES 7 THRU 9
SOURCE WS-DATE

01  TYPE IS CONTROL HEADING WS-LOCATION-CODE.
MOCKUP LINES 10 THRU 13

01  DETAIL-LINE TYPE IS DETAIL.
MOCKUP LINE 14
SOURCE WS-LOCATION-CODE       GROUP INDICATE
SOURCE WS-LAST-COUNT-MONTH    PIC 99
SOURCE WS-LAST-COUNT-DAY      PIC 99
SOURCE WS-LAST-COUNT-YEAR     PIC 99
SOURCE WS-QTY-IN-STOCK
SOURCE WS-QTY-ISSUED
SOURCE WS-QTY-RECEIVED
REFERENCE WS-NO-OF-SALES      PIC zzz9
01  TYPE IS CONTROL FOOTING WS-LOCATION-CODE.
MOCKUP LINES 15 THRU 21
SUM WS-QTY-IN-STOCK
SUM WS-QTY-ISSUED
SUM WS-QTY-RECEIVED
SUM WS-NO-OF-SALES
01  TYPE IS CONTROL FOOTING FINAL.
MOCKUP LINES 22 THRU 30
SUM WS-QTY-IN-STOCK
SUM WS-QTY-ISSUED
SUM WS-QTY-RECEIVED
```

```
SUM WS-NO-OF-SALES

01  TYPE IS PAGE FOOTING.
MOCKUP LINE 31
SOURCE PAGE-COUNTER

01  TYPE IS REPORT FOOTING.
MOCKUP LINES 32 THRU 33

$TP-ENTRY
    OPEN INPUT INPUT-FILE
    ...OUTPUT REPORT-OUTPUT-FILE
    ACCEPT WS-DATE-HOLD FROM DATE
    MOVE WS-DATE-YY-X TO WS-DATE-YY
    MOVE WS-DATE-MM-X TO WS-DATE-MM
    MOVE WS-DATE-DD-X TO WS-DATE-DD
    INITIATE STOCK-REPORT
    REPEAT
        READ INPUT-FILE INTO WS-PART-STOCK-REC
    UNTIL AT END ON INPUT-FILE
        GENERATE DETAIL-LINE
    TERMINATE STOCK-REPORT
    CLOSE INPUT-FILE
    ... REPORT-OUTPUT-FILE
```

Printed report

```
                WONDERFUL WIDGETS INCORPORATED

                        STOCK REPORT

                         01/25/91

                        MID-ATLANTIC
                        STOCK REPORT

LOCATION           LAST COUNT     QUANTITY IN    QUANTITY    QUANTITY
                      DATE          STOCK         ISSUED     RECEIVED

ALEXANDRIA          05/01/90            111          222         333
                    06/01/90            111          222         333
                    07/01/90            111          222         333
                    08/01/90            111          222         333
                    09/01/90            111          222         333
                    10/01/90            111          222         333
                    11/01/90            111          222         333
```

TOTAL BY LOCATION:	777	1,554	2,331

TOTAL NUMBER OF SALES BY LOCATION: 3,108

LOCATION	LAST COUNT DATE	QUANTITY IN STOCK	QUANTITY ISSUED	QUANTITY RECEIVED
BALTIMORE	06/01/90	11	22	33
	08/01/90	11	22	33
	09/01/90	11	22	33
	10/01/90	11	22	33
	11/01/90	11	22	33
TOTAL BY LOCATION:		55	110	165

TOTAL NUMBER OF SALES BY LOCATION: 220

PAGE 1

MID-ATLANTIC
STOCK REPORT
01/25/91

LOCATION	LAST COUNT DATE	QUANTITY IN STOCK	QUANTITY ISSUED	QUANTITY RECEIVED
ROCKVILLE	05/01/90	11	22	33
	06/01/90	11	22	33
	07/01/90	11	22	33
	08/01/90	11	22	33
	09/01/90	11	22	33
	10/01/90	11	22	33
	11/01/90	11	22	33
TOTAL BY LOCATION:		77	154	231

TOTAL NUMBER OF SALES BY LOCATION: 308

LOCATION	LAST COUNT DATE	QUANTITY IN STOCK	QUANTITY ISSUED	QUANTITY RECEIVED
WASHINGTON	06/01/90	111,111	222,222	333,333
	07/01/90	111,111	222,222	333,333
	08/01/90	111,111	222,222	333,333
	09/01/90	111,111	222,222	333,333
	10/01/90	111,111	222,222	333,333
	11/01/90	111,111	222,222	333,333

```
TOTAL BY LOCATION:                  666,666    1,333,332   1,999,998

TOTAL NUMBER OF SALES BY LOCATION:       26,664

                                                          PAGE    2

                          MID-ATLANTIC
                          STOCK REPORT
                           01/25/91

TOTAL WONDERFUL WIDGETS IN STOCK:       667,575
TOTAL WONDERFUL WIDGETS ISSUED:       1,335,150
TOTAL WONDERFUL WIDGETS RECEIVED:     2,002,725
TOTAL WONDERFUL WIDGETS SOLD:            30,300

                                                  PAGE    3
                             *****  END OF REPORT  *****
```

12 Using the APS/ENDEVOR Interface

This chapter contains the following sections:

- *APS/ENDEVOR Overview*
- *Using APS/ENDEVOR*

APS/ENDEVOR Overview

Version control for APS applications

The APS/ENDEVOR Interface is an interface between Micro Focus's APS for z/OS and LEGENT Corporation's ENDEVOR/MVS software management product. The interface lets you manage your APS application components--called elements in ENDEVOR--using ENDEVOR from within APS for z/OS.

APS/ENDEVOR tasks

Specifically, you can do the following:

- Store and retrieve multiple revisions of an APS application component.

  APS/ENDEVOR stores all revisions of an application component in a single controlled member in the ENDEVOR/MVS library. You can retrieve any revision at any time.

- Manage all components of an application as one group.

  For an application component that references all components of an application, APS/ENDEVOR manages all its referenced components as a group. You can add, update or sign in, and retrieve and signout all components of an application with one request.

- Resolve access conflicts.

  When you retrieve a revision to modify it, you can sign it out to prevent other developers from simultaneously changing that revision.

- Display the history of source code changes.

  You can display just the statements that differ between a specific revision and the preceding one, instead of comparing revisions line by line. Alternatively, you can display all the inserted and deleted statements in all revisions of a component.

- Display component information.

  You can display log information on all revisions of a component, including creators and creation dates, the origin of the base revision, and when it was last generated and retrieved, and by whom.

Supported actions and displays

The APS/ENDEVOR Interface lets you use the basic ENDEVOR/MVS action and display functions that you are likely to need. For those functions not supported by the interface, you can invoke ENDEVOR/MVS from within APS. This document discusses only those ENDEVOR/MVS functions relevant to understanding and using the interface.

The APS/ENDEVOR Interface provides options that correspond to ENDEVOR/MVS actions and displays, as follows. Each option works in APS just as it works in ENDEVOR.

Interface Option	ENDEVOR/MVS Function
Checkin	Add/Update or Signin
Checkout	Retrieve and, by default, Signout
Summary Report	Summary Element Display
Master Report	Master Element Display
Browse Report	Browse Element Display
View Differences Report	Changes Element Display
History Report	History Element Display

Checkin

The check in action adds to or updates the ENDEVOR library with an APS component from an APS Project.Group. Alternatively, you can sign in a component at checkin, without adding to or updating the library.

The first time you check in a component from your APS Project.Group, APS/ENDEVOR creates a controlled member that stores the component, and all subsequent revisions of that component, in the ENDEVOR/MVS library. The first checked in component is the first revision. It has the version number number 01 and the level number 00, expressed as 01.00.

A controlled member contains the following information:

- The complete text of the latest revision of the component.
- The modified text, or set of deltas, from all prior revisions. When a user checks out any prior revision, ENDEVOR reconstructs it from the latest revision and the deltas of the prior revisions.
- Log information on all revisions that you can display in reports.

APS submits a batch job to check in the Program (PG) and Screen (SC) component types; for all other component types, APS executes a job immediately.

Checkout The checkout function retrieves and, by default, signs out a revision from a controlled member of the ENDEVOR library to an APS Project.Group so that you can modify it.

By signing out a revision, you prevent anyone else from modifying it. Conversely, you cannot checkout a revision that has been checked out by some one else. You can, however, override the signout, assuming you have authority to do so. Otherwise, the signout is released when the component is moved or transferred to another Stage.

Reports APS/ENDEVOR provides reports that help you monitor the changes made to APS components in the ENDEVOR library.

The View Differences report lets you display the source statements that differ between a specific component revision and the preceding one.

The four View Print reports let you display log and source change information on one or all revisions of a component.

Report	Description
Browse	• Log information on all revisions, including creators; creation dates; number of statements; CCIDs; comments; when the component was last generated and retrieved, and by whom. • All statements in the specified revision, marked with the level number at which they were first inserted.

Report	Description
History	• Log information on all revisions, including creators; creation dates; number of statements; CCIDs; comments; when the component was last generated and retrieved, and by whom. • All inserted and deleted statements that ever existed in all revisions of the component, marked with the level number at which they were inserted or deleted.
Master	• Information on a component, including its processor group; the last action performed against it; its current signout status; when it was last modified and generated, and by whom; the origin of its base revision; who moved or transferred the component from a Stage, and when.
Summary	• Log information on all revisions, including creators; creation dates; number of statements; number of inserted and deleted statements.

For information on ENDEVOR/MVS, see the ENDEVOR/MVS User's Guide and the ENDEVOR/MVS Administrator's Guide.

Using APS/ENDEVOR

This section provides instructions for accessing the interface, specifying the APS location for checkins and checkouts, executing checkins and checkouts, and running reports.

Accessing APS/ENDEVOR Options

1 To access the APS/ENDEVOR Interface, select option 5, Version Control System, from the APS Main Menu. The APS/ENDEVOR Version Control Menu displays.

2 Select one of the five APS/ENDEVOR options.

Option	Function
Project Group Environment	Specify the APS Project and Group for checking components in and out.
Checkin	Add to or update the ENDEVOR library with an APS component from an APS Project.Group, or signin a component without adding to or updating the library.
Checkout	Retrieve and, by default, sign out a revision from a controlled member of the ENDEVOR library to an APS Project.Group so that you can modify it.
View Differences	Display a report showing the changed source statements that differ between a specific component revision and the preceding one.
View Print	Display reports showing log and source change information on one or all revisions of a component.

If you need to use an ENDEVOR/MVS function not presented on the APS/ENDEVOR Menu, you can access ENDEVOR/MVS from any APS screen by entering ndvr in the Command field on any APS screen.

Specifying a Project and Group

1 From the APS/ENDEVOR Version Control Menu, select option 0, Project Group Environment. Alternatively, enter proj in the Command field on any APS screen.

2 Specify the Project and Group you want to check components in from, and check components out to. You can change the value at any time.

Checking a Component In

1 From the APS/ENDEVOR Version Control Menu, select option 1, Checkin. Alternatively, enter ci in the Command field on any APS screen.

2 Complete the Checkin screen fields as follows:

Screen Field	Description
Entity Type	Entity Type of the APS component to check in. Valid values: ap — Application Painter component in APSAPPL plus its related component in APRAPPL cn — Scenario Painter component in APSCNIO ds — Data Structure Painter component in APSDATA ox — Online Express component in APSEXPS pg — Program Painter component in APSPROG plus its related component in APRPROG rp — Report Mock-up Painter component in APSREPT sc — Screen Painter component in APSSCRN For other APS component types in your Project.Group, specify a data set name, such as USERMACS and DDISYMB.
Member	Component name to check in, or leave the Member field blank to select from a member list.
System	ENDEVOR System name, if it differs from the default System name for your current APS Project.Group.
Subsystem	ENDEVOR Subsystem name, if it differs from the default Subsystem name for your current APS Project.Group.
Comment	Text comment for the checkin.
CCID	ENDEVOR CCID for the checkin.
Bypass Gen Processor	Default: No. Specify yes to bypass the associated ENDEVOR Generate Processor.
Delete Input Source	Default: No. Specify yes to delete the component from the APS Project.Group.

Screen Field	Description
Processor Group	Name of the ENDEVOR Processor Group.
Override Signout	Default: No. Specify yes to override an existing signout. You must have authority to do so.
Signin Only	Default: No. Specify yes to Signin only, releasing a previous signout of the component issued with your user ID; the Add or Update action is not executed.
Stage	ENDEVOR Stage number for signin.
Component Parts	To use when checking in AP and PG component type components. Valid values: none — Default. Process only the component specified in the Member field. all — Process the component specified in the Member field and all its associated component parts, or components. list — Display the Component Types Selection screen, to select the associated component types for processing. APS submits a batch job to perform the checkin when some or all component parts are checked in with the component specified in the Member field.

3 Press Enter to execute the checkin.

4 Check the ENDEVOR Action Summary Report to ensure that the checkin job succeeded.

Checking a Revision Out

1 From the APS/ENDEVOR Version Control Menu, select option 2, Checkout. Alternatively, enter co in the Command field on any APS screen.

2 Complete the Checkout screen fields as follows.

Screen Field	Description
Entity Type	Component Type of the component to checkout. Valid values same as for Checkin.
Member	Member name to checkout, or leave the Member field blank to select from a member list.
System	ENDEVOR System name, if it differs from the default System name for your current APS Project.Group.
Subsystem	ENDEVOR Subsystem name, if it differs from the default Subsystem name for your current APS Project.Group.
Stage	ENDEVOR Stage number of the member to checkout.
Version	Defaults to the current revision. You can optionally override this value with another version number.
Level	Defaults to the current level. You can optionally override this value with another level number.
Comment	Text comment for the checkout.
CCID	ENDEVOR CCID to associate with the checkout.
No Signout	Specify yes to checkout and browse the member without signing it out to your user ID.
Replace Member	Specify yes to overlay an existing member in the APS Project.Group.
Override Signout	Specify yes to override an existing Signout by another user. You must have authority to do so.
Component Parts	For use when checking out AP and PG component type components. Valid values same as for checkin.

3 Press Enter to execute the checkout.

4 Check the ENDEVOR Action Summary Report to ensure that the checkout job has succeeded.

Running the View Differences Report

1 On the APS/ENDEVOR Version Control Menu, select option 3, View Differences. Alternatively, enter df in the Command field on any APS screen.

2 Specify the name, location, version, and level of a controlled member revision you want to report on. The report defaults to the current version and level; you can override with any version and level.

Running the View Print Reports

1 On the APS/ENDEVOR Version Control Menu, select option 4, View Print. Alternatively, enter vp in the Command field on any APS screen.

2 Run any of the reports by specifying the report, name, and location of a controlled member you want to report on. For reports on a specific revision, specify the version and level of the revision to report on. The report defaults to the current version and level; you can override with any version and level.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X